NATIONAL OPEN UNIVERSITY OF NIGERIA

COURSE CODE: CIT 905

COURSE TITLE:

ADVANCED DATABASE MANAGEMENT SYSTEM

COURSE GIUDE

ADVANCED DATABASE MANAGEMENT SYSTEM

Course Team

Course Code	CIT 905
Course Title	**Advanced Database Management System**
Course Developer/Writer	Dr. Udoinyang G. Inyang Department of Computer Science University of Uyo Nigeria
Content Editor	Prof. Okonkwo Obikwelu Raphael Nnamdi Azikiwe University, Awka
Course Material Coordination	Dr. Vivian Nwaocha, Dr. Greg Onwodi & Dr. Francis B.Osang Computer Science Department National Open University of Nigeria

NATIONAL OPEN UNIVERSITY OF NIGERIA

CIT 905 : ADVANCED DATABASE MANAGEMENT SYSTEM

National Open University of Nigeria
Headquarters
Headquarters

Plot 91, Cadastral Zone, Nnamdi Azikiwe Expressway, Jabi – Abuja,

Nigeria

Lagos Office

14/16 Ahmadu Bello Way

Victoria Island

Lagos

e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

Published by
National Open University of Nigeria
Printed 2020
ISBN: XXX-XXX

CONTENTS PAGE

Introduction

The course, Advanced Database Management System, is a 2 credit hour core course for students studying towards acquiring the Doctor of Philosophy (PhD) in in Information Technology. In this course we will study about the Database Management System as a key role in Information Management. Various principles of database management system (DBMS) as well as its advanced features are discussed in this course. This course also considers distributed databases and emerging trends in database system.

The overall aim of this course is to introduce you to various ways of designing and implementing database systems, features and distributed databases. In structuring this course, we commence with the basic design and implementation of relational databases. There are four modules in this course, each module consists of units of topics that you are expected to complete in 2 hours. The four modules and their units are listed below.

What You Will Learn in this Course

The overall aims and objectives of this course provide guidance on what you should be achieving in the course of your studies. Each unit also has its own unit objectives which state specifically what you should be achieving in the corresponding unit. To evaluate your progress continuously, you are expected to refer to the overall course aims and objectives as well as the corresponding unit objectives upon the completion of each.

Course Aims

The overall aims and objectives of this course will help you to:

1. Develop your knowledge and understanding of the underlying principles of Relational Database Management System
2. Build up your capacity to learn Database Management System advanced features
3. Develop your competence in enhancing database models using distributed databases
4. Build up your capacity to implement and maintain an efficient database system using emerging technologies and tools.

Course Objectives

Upon completion of the course, you should be able to:

1. Describe the basic concepts of Relational Database Design
2. Explain Database implementation and tools
3. Describe SQL and Database System catalog.
4. Describe the process of DB Query processing and evaluation.
5. Discuss the concepts of transaction management.
6. Explain the Database Security and Authorization.
7. Describe the design of Distributed Databases.
8. Know how to design a Database and XML.
9. Describe the basic concept of Data warehousing and Data mining
10. Discuss the emerging Database Models, Technologies and Applications

Working through this Course

We designed this course in a systematic way, so you need to work through it from Module one, Unit One (1) through to Module four, Unit five (5). This will enable you appreciate the course better.

Course Materials

Basically, we made use of textbooks and online materials. You are expected to I, search for more literature and web references for further understanding. Each unit has references and web references that were used to develop them.

Online Materials

Feel free to refer to the web sites provided for all the online reference materials required in this course. The website is designed to integrate with the print-based course materials. The structure follows the structure of the units and all the reading and activity numbers are the same in both media.

Study Units

Course Guide

Module 1: Database Design and Implemental

Unit 1: Concepts of Relational Database

Unit 2: Database Design and Implementation

Unit 3: Advance SQL

Unit 4: Database System Catalog

Module 2: DBMS Advance Features

Unit 1: Query Processing & Evaluation

Unit 2: Transaction Management and Recovery

Unit 3: Database Security & Authorization

Module 3: Distributed Databases

Unit 1: Distributed database System

Unit 2: Enhanced Database Models

Unit 3: Object Oriented Database

Unit 4: Database and XML

Unit 5: Introduction to Data Warehousing

Unit 6: Introduction to Data Mining

Module 4: Emerging Trends and Example of DBMS Architecture

Unit 1: Relational and Non-Relational databases

Unit 2: Conventional Database Management Systems

Unit 3: Emerging Database Systems

Unit 4. Database Services and Service Providers

Unit 5: Modern Database Applications

Module one describes Database Design and Implementation.

Module Two explains the DBMS advanced features.

Module Three discusses the Distributed Database.

Module Four discusses emerging trends in DBMS including technologies and applications.

Equipment

In order to get the most from this course, it is essential that you make use of a computer system which has internet access.

Recommended System Specifications:

Processor
2.5 GHZ Intel compatible processor
4GB RAM
500 GB hard drive with 5 GB free disk
CD-RW drive.
TCP/IP (installed)

Operating System
Microsoft Windows 10
Microsoft office 2007
Antivirus

Monitor*
21-inch
1024 X 768 Resolution
16-bit high color
*Non Standard resolutions (for example, some laptops) are not supported.

DBMS Tools
ORACLE
PostgreSQL

Hardware
Open Serial Port (for scanner)
120W Speakers

Hardware is constantly changing and improving, causing older technology to become obsolete. An investment in advanced and more efficient technology will more than pay for itself in improved performance results.

If your system does not meet the recommended specifications, you may experience considerably slower processing when working in the application. Systems that exceed the recommended specifications will provide better handling of database files and faster processing time, thereby significantly increasing your productivity

Assessment

The course, Advanced Database Management Systems entails attending a two-hour final examination which contributes 50% to your final grading. The final examination covers materials from all parts of the course with a style similar to the Tutor- marked assignments.

The examination aims at testing your ability to apply the knowledge you have learned throughout the course, rather than your ability to memorize the materials. In preparing for the examination, it is essential that you receive the activities and Tutor-marked assignments you have completed in each unit. The other 50% will account for all the TMA's at the end of each unit.

Tutor-Marked Assignment

About 20 hours of tutorials will be provided in support of this course. You will be notified of the dates, time and location for these tutorials, together with the name and phone number of your tutor as soon as you are allotted a tutorial group.

Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assistance to you during the course. You must mail your TMAs to your tutor well before the due date (at least two working days are required). They will be marked by your tutor and returned to you as soon as possible.

Do not hesitate to contact your tutor by phone, e-mail if you need help. The following might be circumstances in which you would find help necessary. You can also contact your tutor if:

1. you do not understand any part of the study units or the assigned readings
2. you have difficulty with the TMAs
3. you have a question or problem with your tutor's comments on an assignment or
4. with the grading of an assignment

You should try your best to attend tutorials, since it is the only opportunity to have an interaction with your tutor and to ask questions which are answered instantly. You can raise any problem encountered in the course of your study. To gain maximum benefit from the course tutorials, you are advised to prepare a list of questions before attending the tutorial. You will learn a lot from participating in discussions actively.

Course Overview

This section proposes the number of weeks that you are expected to spend on the four modules comprising of 30 units and the assignments that follow each of the unit. We recommend that each unit with its associated TMA is completed in one week, bringing your study period to a maximum of 30 weeks.

How to Get the Most from this Course

In order for you to learn various concepts in this course, it is essential to practice. Independent activities and case activities which are based on a particular scenario are presented in the units. The activities include open questions to promote discussion on the relevant topics, questions with standard answers and program demonstrations on the concepts. You may try to delve into each unit adopting the following steps:

i. Read the study unit
ii. Read the textbook, printed or online references
iii. Perform the activities
iv. Participate in group discussions
v. Complete the tutor-marked assignments
vi. Participate in online discussions

This course makes intensive use of materials on the world-wide web Specific web address will be given for your reference. There are also optional readings in the units. You may wish to read these to extend your knowledge beyond the required materials. They will not be assessed.

Summary:

The course, Advanced Database Management Systems is intended to develop your understanding of the basic concepts of database systems, thus enabling you acquire skills in designing and implementing Database Management Systems. This course also provides you with practical knowledge and hands-on experience in implementing and maintaining a system. We hope that you will find the course enlightening and that you will find it both interesting and useful. In the longer term, we hope you will get acquainted with the National Open University of Nigeria and we wish you every success in your future

MAIN COURSE

ADVANCED DATABASE MANAGEMENT SYSTEM

Course Developer: Dr. Udoinyang G. Inyang
Department of Computer Science
University of Uyo
Uyo, Nigeria

Course Writer : Dr. Udoinyang G. Inyang
Department of Computer Science
University of Uyo
Uyo, Nigeria

Programme Leader :
National Open University of Nigeria

Course Coordinator :
National Open University of Nigeria

NATIONAL OPEN UNIVERSITY OF NIGERIA

CIT 905 : ADVANCED DATABASE MANAGEMENT SYSTEM

National Open University of Nigeria
Headquarters
No. 5 Dares Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja
Nigeria
e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

Lagos Office:
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Published by
National Open University of Nigeria
Printed 2020
ISBN: XXX-XXX

CONTENTS PAGE

Unit 2: Transaction Management and Recovery

Unit 3: Database Security & Authorization

Unit 3: Object Oriented Database

Unit 4: Database and XML

Unit 5: Introduction to Data Warehousing

Unit 6: Introduction to Data Mining

Module 4: Emerging Trends and Example of DBMS Architecture

Unit 1: Relational and Non-Relational databases

Unit 2: Conventional Database Management Systems

UNIT 3: Emerging Database Systems

Blank Page

MODULE 1: DATABASE DESIGN AND IMPLEMENTATION

UNIT 1: RELATIONAL DATABASE DESIGN AND IMPLEMENTATION

1. INTRODUCTION

This unit discusses the relational data model. Which uses the concept of mathematical relation, which looks somewhat like a table of values, as its basic building blocks, and has its theoretical basics in *set theory* and *first order predicate logic*. In this unit, we will discuss the basic characteristics for the relational model and its normalization processes. The model has been implemented in a large number of commercial systems over the last years.

1.1 OBJECTIVES.

The objectives of this unit include to:

1. Describe the data model and its concepts.
2. discuss the relational database model.
3. Help the students understand the concepts of normalization in database design

1.2 What is a data model?

A data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. In this part, we focus on the relational model. It is a notation for describing data or information. The description generally consists of three parts: Structure of the data, Operations on the data and Constraints on the data. In unit we describe the relational database model

1.3 What Is Relational Database?

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. It also includes a Data Manipulation Language (DML) and Data Definition Language (DDL). The relational model is today the primary data model for commercial data processing applications - especially for storing financial records, manufacturing and logistical

information, personnel data and much more. It attained its primary position because of its simplicity, which eases the job of the programmer, compared to earlier data models such as the network model or the hierarchical model.

1.2.1 Structure of Relational Databases

A relational database consists of a collection of **tables**, each of which is assigned a unique name. For example, consider the *Facilitator* table (Table 2.1), which stores information about facilitators. The table has four columns: *ID*, *name*, *department* and *rank*. Each row of this table records information about a facilitator, consisting of the facilitator's *ID*, *name*, *department*, and *rank*. Similarly, the *course* table (Table 2.2) stores information about courses, consisting of a *course code*, *title*, *department*, and *Credit Hour(CH)*, for each course. Note that each instructor is identified by the value of the column *ID*, while each course is identified by the value of the column *course code*.

Table 2.1: Facilitator table

ID	Name	Department	Rank
001010	Patience	Computer Science	Reader
001023	Inyang	Chemistry	Professor
001058	Godwin	Physics	Lecturer 1
002010	Daniel	History	Professor
003010	Suleman	Geography	Reader

Table 2.2: Facilitator table

Course Code	Title	Department	CH
CIT 751	Patience	Computer Science	3
CHM 222	Inyang	Chemistry	4
PHY 234	Godwin	Physics	3
HIS 442	Daniel	History	2
GPY 130	Suleman	Geography	4

In general, a row in a table represents a *relationship* among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of *table* and the mathematical concept of *relation*, from which the relational data model takes its name. In mathematical terminology, a *tuple* is simply a sequence (or list) of values. A relationship between *n* values is represented mathematically by an *n-tuple* of values, i.e., a tuple with *n* values, which corresponds to a row in a table.

Thus, in the relational model the term relation is used to refer to a table, while term tuple is used to refer to a row. Similarly, the term attribute refers to a column header of a table while the data type describing the types of values that can appear in each column is called a domain. Examining Table 2.1, we can see that the relation instructor has four attributes: ID, name, department, and rank. We use the term **relation instance** to refer to a specific instance of a relation that is, containing a specific set of rows. The instance of a *facilitator* shown in Table 2.1 has 5 tuples, corresponding to five facilitators.

Each attribute of a relation, has a set of permitted values, called the **domain** of that attribute. Thus, the domain of the *rank* attribute of relation is the set of all possible ranks values, while the domain of the *name* attribute is the set of all possible *facilitators'* names. A domain is **atomic** if elements of the domain are considered to be indivisible units. For example, suppose the table *facilitator* had an attribute *phone number*, which can store a set of phone numbers associated to the *facilitator*. Then the domain of *phone number* would not be atomic, since an element of the domain is a set of phone numbers (may be more than one phone numbers), and it has subparts, namely each individual phone numbers in the set.

The important issue is not what the domain itself is, but rather how we use domain elements in our database. Suppose now that the *phone number* attribute stores a single phone number. Even then, if we split the value from the phone number attribute (for example +234-873-424-1626), into a country code, network provider, an area code and a local number, then it is considered as a non-atomic value. If view each phone number as a single indivisible unit, then the attribute *phone number* would then have an atomic domain.

1.2.2 Database Schema

A **database schema**, is the logical design of the database, and the **database instance**, is a snapshot of the data in the database at a given instant in time. A database schema represents the logical configuration of all or part of a relational database. It can exist both as a visual representation and as a set of formulas (known as integrity constraints), that govern a database. These formulas are expressed in a DDL, such as SQL. As part of a data dictionary, a database schema indicates how the entities that make up the database relate to one another, including tables, views, stored procedures, and more. At the most basic level, a database schema indicates which tables or relations make up the database, as well as the attributes included on each table. Thus, the terms schema diagram and entity-relationship diagram are often interchangeable. The concept of a relation corresponds to the programming-language notion of a variable, while the concept of a **relation schema** corresponds to the programming-language notion of type definition.

A database designer creates a database schema mainly to help programmers whose software will interact with the database. The process of creating a database schema is called **data modeling**. There are two main kinds of database schema:

1. A logical database schema conveys the logical constraints that apply to the stored data. It may define integrity constraints, views, and tables.
2. A physical database schema lays out how data is stored physically on a storage system in terms of files and indices.

1.2.3 What is Database instance or database schema?

These terms, though related, do not mean the same thing. A database schema is a sketch of a planned database. It does not actually have any data in it. A database instance, on the other hand, is a snapshot of a database as it existed at a particular time. Thus, database instances can change over time, whereas a database schema is usually static, since it is arduous task to change the structure of a database once it is operational. Database schemas and database instances can affect one another through a database

management system (DBMS). The DBMS makes sure that every database instance complies with the constraints imposed by the database designers in the database schema.

1.2.3 Database Schema Integration

Schema integration describes the task of building a global data schema from a set of local schemas (usually with overlapping semantics) to provide the user with a unified view of the entire dataset. Schema integration is defined as the process of merging several conceptual schemas into a global conceptual schema that represents all the requirements of the application. Schema integration is used to merge two or more database schemas into a single schema that can store data from both the original databases.

The goal is to give the user of the global schema an illusion of a single dataset specified by a single schema.

1. Schema integration is used when two or more existing databases must be combined, for example, when a new management information system is being developed.
2. Schema integration may be used when the process of database design is too large to be carried out by one individual. Two or more designers will build models of different parts of the database and use schema integration to merge the resulting models.

There are two major types of schema integration:

1. **View Integration:** View integration takes place during the design of a new database when user requirements may be different for each user group. View integration is used to merge different viewpoints into a single data model.
2. **Database Integration:** Database integration is used when two or more databases must be combined to produce a single schema, called a global schema.

Reasons for Schema Integration

1. The structure of the database for large applications is too complex to be modeled by a single designer in a single view.
2. User groups typically operate independently in organization and have their own requirements and expectations of data, which may conflict with other user groups
3. When two database schemas are designed by different designers using different user requirements, the resulting schemas will often present contrasting views of the same data.

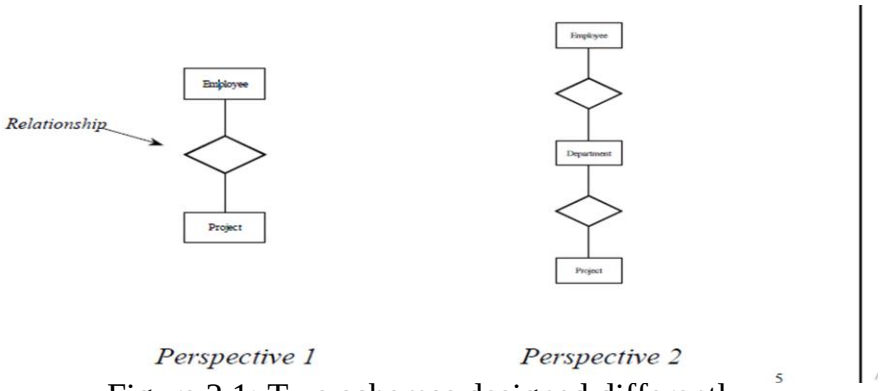


Figure 2.1: Two schemas designed differently

For example above, the relationship between employee and project in one database is represented as a relationship between employee, department and project in another database.

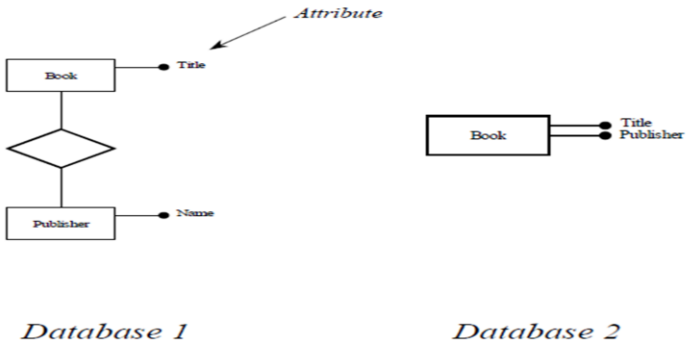


Figure 2.2: entity relationship in two databases

This situation might occur in an organization that allows different departments to have different rules as to how employees are allocated to projects. For example, in one department employees may be assigned to projects while in another department employees may not be considered to be directly related to a project.

Different databases may treat the same concepts in different ways.

In the above example, the publisher concept is an entity in database 1 but an attribute in database 2. There are two situations that must be dealt with during schema integration:

1. When different concepts are modelled in the same way. For example, in a university database staff and students may be represented by the entity person even though they are different concepts.
2. When the same concepts are modelled in different way. For instance, the above example models the concept of a publisher as an entity and as an attribute.

Incompatible designs:

1. Two database designs may be incompatible because mistakes were made in the initial design or there are different constraints placed on the data. For instance, in the above example, the relationship between employee and project is represented as a one-to-many relationship in database 1 and as a many-to-many relationship in database 2.
2. This problem may be caused by mistakes made during the initial database analysis task or because users of the system have different working practices. For example, one department in an organization, which works on small projects, may allocate one employee to a project but a different department, which works on large projects, may allocate many employees to a project.

During schema integration these different viewpoints must be reconciled

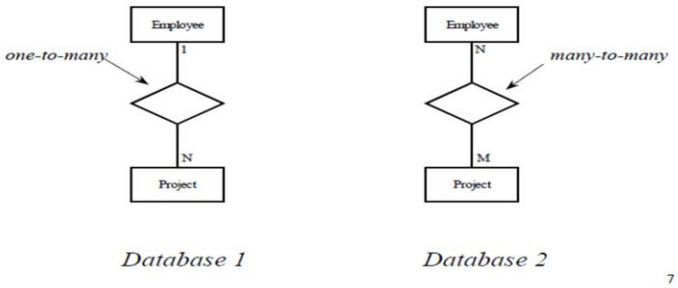


Figure 2.3: schemas reconciliation

Steps and goals of the integration process

i) Pre-integration.

1. Choose integration processing strategies
2. This governs the choice of schemas to be integrated

ii) Comparison of the schemas.

Schemas are analyzed and compared to determine the correspondences among concepts and detect possible conflicts.

iii) Conforming the schemas.

Once conflicts are detected, an effort is made to resolve them so that the merging of various schemas is possible.

Automatic conflict resolution is generally not feasible; interaction with designers is required.

iv) Merging and Restructuring.

1. The schemas are ready to be superimposed, giving rise to some intermediate integrated schema(s).
2. The second strategy for schema integration is to integrate some of the schemas (e.g. two) and then to integrate the resulting schemas.
3. This approach would be more appropriate when the schemas are complex or when there are a large number of schemas

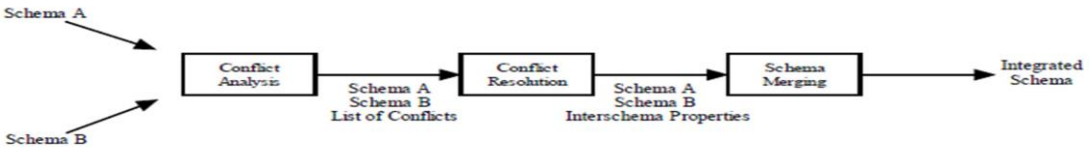


Figure 2.4: schema integration process

The schema integration process starts with two or more schemas and involves three main stages:

1. **Conflict Analysis**: during conflict analysis differences in the schemas are identified, for example, similar concepts that are represented in different ways. Check all conflicts in the representation of the same objects in different schemas.

2. **Conflict Resolution:** resolution the conflicts identified during conflict analysis are resolved. For example, a common method of representing equivalent concepts will be decided upon. This process may involve discussing the problems with the users or correcting errors in the schemas.
3. **Schema merging:** During schema merging the schemas are merged into a single schema using the decisions made during the conflict resolution.

Types of Conflicts

1. **Naming Conflicts:** It occurs when different names are used for the same attribute or the same object are given different names in their respective databases. Different naming conventions are used by databases for objects.
1. **Structural conflicts.** This type of conflict occurs when the actual method of representing the same concept in different databases is different or incompatible.

Types of Name conflicts :

1. **Synonyms:** Objects that are the same but have different names. For example, two public transport databases may have entities called passenger and customer, SNO and S#, EMP and EMPLOYEE, TAKE and ENROL. These entities may be the same entity.
2. **Homonyms:** Concepts or objects that are different but have the same names.. For example, two publishing databases may have entities called publication but in one database a publication may be a book while in the other database a publication may be a journal.
3. **Name conflicts** cause a problem because information may be duplicated in the integrated database. It is important to identify those data items in each schema that actually represents the same concept or that should be represented using different structures in the integrated schema.
4. **Synonyms** may be removed from the database by renaming the concepts so that they have the same name.
5. **Homonyms** may be removed from the database by renaming the concepts so that they have different names.

6. It may be possible to use a superclass/subclass relationship to avoid synonyms or homonyms.

Types of Structural Conflicts: There are three cases

1. **Identical Concepts:** When the same concept in different databases is represented in the same way they may be merged. For example, when an entity publication has the same structure and means the same in two databases the entities may be merged.
2. **Compatible Concepts:** When the same concept in different databases is represented in compatible ways they may be merged. For example, when an entity publication is represented by an attribute in one database and an entity in another database they may be merged by converting the attribute into an entity.
3. **Incompatible Concepts:** When the same concept in different databases is represented using different structures then it may be difficult to merge them directly. For example: Relationships may have different cardinalities (that is, one-to-many and many-to-many). This is also referred to as **cardinality conflicts** of attributes (that is different cardinalities for the same attributes of an entity types in more than one views).
4. Primary keys may be different, that is, Different keys are assigned as identifiers of the same entity type in different schemas.
5. Set relationships may be reversed (e.g. projects contain programmes and programmes contain projects).
6. Incompatible designs must be resolved by re-analyzing the data and adapting one or more of the schemas or by constructing a new, common representation.

Data type conflicts

The same attribute with different data types. For example, integer & real ⇒ integer e.g. char (20) and char (30) ⇒ char (20) e.g. different range values positive integer & [1,100] ⇒ [1,100]

1.1.4 Database Design

In this section, the design issues regarding relational databases are described. In general, the goal of a relational database design is to generate a set of relation schemas that allow us to store information without unnecessary redundancy, yet allowing us to retrieve information easily. A well-structured and efficient database has the following advantages:

1. Saves disk space by eliminating redundant data.
2. Maintains data accuracy and integrity.
3. Provides access to the data in useful ways.

Designing an efficient, useful database is a matter of following the proper process, including these phases:

1. Strategy and planning
2. Requirements analysis
3. Design
4. Development
5. Deployment/implementation
6. Operations and maintenance.

1. **Strategy and planning** – typically the cycle starts with the strategy and planning phase to identify the need and scope of a new system.
2. **Specification Requirements analysis phase** – a more detailed requirements analysis will be carried out which will include identifying what the users require of the system; this will involve conceptual analysis.
3. **Design phase** – this will involve producing a conceptual, logical and physical design. To undertake these processes it is important to be able to understand and apply the data modelling techniques which are covered in this book. When a suitable logical design has been obtained the development phase can begin.
4. **Development phase** – this involves creating the database structure using an appropriate Database

5. **Management System** (DBMS) and usually includes the development of applications that provide a user interface consisting of forms and reports which will allow controlled access to the data held in the database.
6. **Deployment/implementation** – when the system has been developed it will be tested, it will then be deployed ready for use.
7. **Operations and maintenance** – following the system release for use it will be maintained until it reaches the end of its useful life, at this stage the development lifecycle may restart.

1.1.5 Specification Requirements Gathering

The most critical aspect of specification is the gathering and compilation of system and user requirements. This process is normally done in conjunction with managers and users. The initial phase of database design is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements. While there are techniques for diagrammatically representing user requirements, in this unit we restrict ourselves to textual descriptions of user requirements

The major goals in requirements gathering is to:

- collect the data used by the organization,
- identify relationships in the data,
- identify future data needs, and
- determine how the data is used and generated.

The starting place for data collection is gathering existing forms and reviewing policies and systems. Then, ask users what the data means, and determine their daily processes. These things are especially critical:

- Identification of unique fields (keys)
- Data dependencies, relationships, and constraints (high-level)
- The data sizes and their growth rates

Fact-finding is using interviews and questionnaires to collect facts about systems, requirements, and preferences. Five fact-finding techniques:

1. examining documentation (example invoices, invoices, timesheets, surveys etc.), Comb through any existing data systems (including physical and digital files)
2. interviewing
3. observing the enterprise in operation
4. research
5. questionnaires
6. Start by gathering any existing data that will be included in the database. Then list the types of data you want to store and the entities, or people, things, locations, and events, that those data describe, for example:

1. Customers
2. Name
3. Address
4. City, State, Zip
5. Email address
6. Products
7. Name
8. Price
9. Quantity in stock
10. Quantity on order
11. Orders
12. Order ID
13. Sales representative
14. Date
15. Product(s)
16. Quantity
17. Price
18. Total

This information will later become part of the data dictionary, which outlines the tables and fields within the database. Be sure to break down the information into the smallest useful pieces. For instance, consider separating the street address from the country so that you can later filter individuals by their country of residence. Also, avoid placing the same data point in more than one table, which adds unnecessary complexity.

The result of this step is concisely written as a set of users' requirements. These requirements should be specified as detailed and complete a form as possible. In parallel with specifying the data requirements, it is useful to specify the known functional requirements of the application. These consist of the user–defined operations (or transactions) that will be applied to the database and they include both retrievals and updates. In software design, it is common to use data flow diagrams, sequence diagrams, scenarios, and other techniques for specifying functional requirements

1.1.6. Database Design Phase

The requirements gathering and specification provides you with a high-level understanding of the organization, its data, and the processes that you must model in the database. Database design involves constructing a suitable model of this information. Since the design process is complicated, especially for large databases, database design is divided into three phases:

- Conceptual database design
- Logical database design
- Physical database design

1. Conceptual Schema Modelling

Once all the requirements have been collected and analyzed, the next steps is to create a conceptual schema for the database, using a high-level conceptual data model. That is to develop layout or a visual representation of the proposed database.

Why do you need to a Conceptual model?

In many environments modelling is used to ensure that a product will satisfy the user's requirements before it is produced. For example, an architect may use a scale model of a building so the client can see what it will look like before it is built. This allows for any changes to be made to the design following feedback and before any expensive building

work takes place. Similarly, a modelling approach is needed when designing a database system so that interested parties can check that the design will satisfy the requirements.

How can a database system be modelled?

In order to design an effective database system you need to be able to understand an organization's information needs and, in particular, identify the data needed to satisfy these needs. Entity Relationship is an important top-down analysis technique which is used to show the structure of the data used by a system. Initially, a conceptual model is produced which is independent of any hardware or DBMS system; this is achieved by using an Entity Relationship Diagram (ERD) or alternatively a UML Class Diagram (CD). This modelling technique will be used to determine how this business data is structured and show the relationships between the different data entities. The model forms the basis for the design of the database system that will be built.

1.1.7 The Entity-Relationship Model

The entity-relationship (E-R) data model was developed to facilitate database design by allowing specification of an enterprise schema that represents the overall logical structure of a database. The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. Because of this usefulness, many database-design tools draw on concepts from the E-R model.

1.1.7.1 The Entity Relationship Diagram (ERD)

The Entity Relationship Diagram (ERD) shows "entities" and the "relationships" that link them. The entities represent the data items needed by the system and the relationships show how the entities are related to one another. An "entity" is formally called an "entity type" and can be defined as: "A group of objects with the same properties which are identified by the enterprise as having an independent existence'

1.1.7.2 Entity types

In order to produce an ERD you need to identify all the entity types that are relevant to the system being modelled. Do not confuse an entity type with the occurrence of an

entity. Often many entities can be identified, although they are not always relevant to the needs of the system being considered, so care needs to be taken to ensure that only those that are needed are added to the ERD. The following are examples of typical entity types:

For a business system: CUSTOMER, ORDER, INVOICE.

For a university system: STUDENT, LECTURER, COURSE

Entities often fall into one of the following categories:

1. Physical – CAR, BUILDING
2. Human – CUSTOMER, EMPLOYEE
3. Place – FACTORY, SCHOOL
4. Group – DEPARTMENT, TEAM
5. Document – INVOICE, PAYSLIP

When you have identified the entity types, these need to be added to the Entity Relationship Diagram (ERD). Although ERDs can be drawn by hand, it is good practice to use a Computer Aided Software Engineering (CASE) tool to ensure your models can be amended easily and presented in a professional form to others. There are many CASE tools available to support modelling.

The E-R data model employs three basic concepts: entity sets, relationship sets, and attributes.

1.1.7.3 Entity Sets

An entity is a "thing" or "object" in the real world that is distinguishable from all other objects. For example, each person in a university is an entity. An entity has a set of properties, and the values for some set of properties may uniquely identify an entity. For instance, a person may have a person id property whose value uniquely identifies that person. Thus, the value 677-89-9011 for person ID would uniquely identify one particular person in the university. Similarly, courses can be thought of as entities, and course ID uniquely identifies a course entity in the university. An entity may be **concrete**, such as

a person or a book, or it may be abstract, such as a course, a course offering, or a flight reservation.

1. An **entity set** is a set of entities of the same type that share the same properties, or attributes. The set of all people who are instructors at a given university, for example, can be defined as the entity set instructor. Similarly, the entity set student might represent the set of all students in the university.
2. In the process of modeling, we often use the term entity set in the abstract, without referring to a particular set of individual entities. We use the term **extension of the entity** set to refer to the actual collection of entities belonging to the entity set. Thus, the set of actual instructors in the university forms the extension of the entity set instructor.
3. An entity is represented by a set of **attributes**. Attributes are descriptive properties possessed by each member of an entity set. The designation of an attribute for an entity set expresses that the database stores similar information concerning each entity in the entity set; however, each entity may have its own value for each attribute. Possible attributes of the *instructor* entity set are *ID*, *name*, *dept name*, and *salary*. In real life, there would be further attributes, such as street number, apartment number, state, postal code, and country, but we omit them to keep our examples simple. Possible attributes of the *course* entity set are *course ID*, *title*, *dept name*, and *credits*.
4. Each entity has a **value** for each of its attributes. For instance, a particular *instructor* entity may have the value 12121 for *ID*, the value for *name*, the value Finance for *dept name*, and the value 90000 for *salary*.
5. The ID attribute is used to identify instructors uniquely, since there may be more than one instructor with the same name. In the United States, many enterprises find it convenient to use the social-security number of a person2 as an attribute whose value uniquely identifies the person. In general the enterprise would have to create and assign a unique identifier for each instructor.
6. A database for a university may include a number of other entity sets. For example, in addition to keeping track of instructors and students, the university

also has information about courses, which are represented by the entity set course.

1.1.7.4 *Entity selection and validation*

In order to produce the ERD you need to ensure you have identified the entities that are suitable for inclusion. The entities initially selected are usually referred to as **"candidate entities"** as not all may be suitable for inclusion. Entity names are normally **nouns** not verbs. The candidate entities are usually identified by referring to a written system description, a set of requirements, or perhaps the notes from a discussion with a person who has knowledge of the system under consideration.

These nouns will form the candidate entity list.

To ensure that a candidate entity is valid for inclusion on the ERD it should satisfy the following three checks:

1. **It should not be the name of the system being modelled**

   It is a common mistake to include an entity which has the name of the system or organization that is being modelled. For example, if you were producing a model of "NOUN University" it would not be appropriate to include an entity type called NOUN UNIVERSITY or even UNIVERSITY as there is only one occurrence of this university. The whole model would, in reality, represent the university. However, if you were modelling a system that needed to hold data for more than one university, then you would need to include an entity type called UNIVERSITY.

2. **The object should be of importance to the system being studied.**

   There are likely to be many objects in the system being studied but you have to decide whether the object is relevant. This usually means determining if the system users are likely to need to retrieve information about the object. For example, if you were designing a university student information system is a "litter bin" likely to satisfy the check? The answer would be no, but are there any circumstances in which it might? If the purpose of the system was to record all university assets, then

you might need to record information about the litter bins. In that case you would need an entity type to represent this information, though the entity type would be called ASSET and bin would be an entity occurrence.

3. **There should be data attributes that can be associated with the entity**
   There must be at least two attributes for an entity type. If you cannot identify any or only one attribute for the entity then you may need to consider whether, in fact, it is actually an attribute of another entity type.

1.1.7.5 Validating the model

The model should be checked with the client or system users to ensure that all relevant entities have been identified, along with the required attributes. This process may need to be repeated a number of times until everyone is satisfied that all requirements have been met.

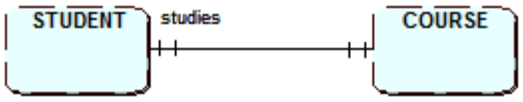
1.1.7.6 Entity Relationships

In order to see what is meant by a relationship, consider the following example which uses the music system entity Types COMPANY, CD, TRACK and CATEGORY.
There are a number of relationships between these entity types as follows:

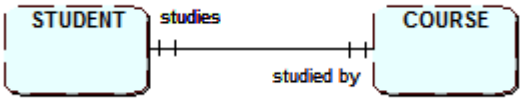
-- A COMPANY produces CDs
-- A CD contains music TRACKs
-- A TRACK belongs to a music CATEGORY

To help you understand the nature of a relationship you may initially find it helpful to see the entity occurrences in a graphical format. If you consider the relationship "COMPANY produces a CD" the diagram below shows how one occurrence of COMPANY relates to two occurrences of CD when looked at from the viewpoint of the COMPANY, which is at the one end of the relationship. From this direction the relationship can be read as "A COMPANY produces CDs."

If there is a relationship between an occurrence of one entity type and an occurrence of another entity type, then it is shown on the entity relationship diagram as a line linking the two entity symbols. The relationship between the two entities should be labelled by using a suitable verb. For example the relationship "STUDENT Studies a COURSE" would be represented as follows



As it is important to consider a relationship from both directions you should also label the relationship from COURSE to STUDENT as follows:



The relationship labels should be positioned as above, near to the relevant entities to aid readability.

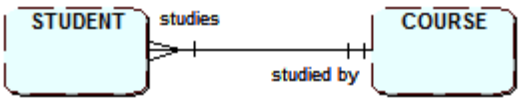
Relationship cardinality

Once you have established a relationship between two entity types it is important to consider how many occurrences of one entity could be related to the other entity. This is referred to as "cardinality".

There are three types of relationship cardinality:

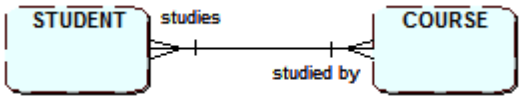
1. One to One (abbreviated as 1:1)
2. One to many (abbreviated as 1:M)
3. Many to Many (abbreviated as M:M or M:N)

Using the earlier example of the relationship between STUDENT and COURSE, consider the relationship from the STUDENT's viewpoint. A student can study a course and if you then consider the relationship from the COURSE viewpoint, you can say that a COURSE can be studied by many STUDENTs. This would be a **"one to many"** (1:M) relationship and would be drawn on the ERD as follows:

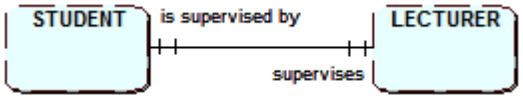


The "crow's foot" symbol is used to represent many and is placed at the "many" end of the relationship. The relationship would be read formally as "a student studies one and only one course and a course is studied by one or many students". If you now reconsider the relationship between STUDENT and COURSE but want to be able to show that a student may study more than one course, you now need to

alter the relationship to show as a "many to many" (M:M or M:N). A M:N relationship is sometimes written as M:M though M:N is preferred so as to indicate that the number of occurrences at one end of the relationship can be different from number at the other end of the relationship. This is drawn on the ERD as follows:



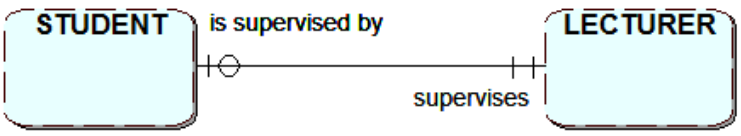
The final cardinality type that needs to be examined is for the "one to one" relationship. If a STUDENT is assigned a LECTURER as a supervisor and the LECTURER only supervises one student, you can show this as follows on the ERD: In this case the relationship would be read as "a student is supervised by one and only one lecturer and a lecturer supervises one and only one student



1.1.7.5 Relationship optionality

When describing an entity relationship you need to record the fact on the ERD that in some cases an occurrence of an entity type may not always be present, in which case the relationship is said to be **optional**. Using the previous cardinality example, the model states that a lecturer supervises a student. However, what if some lecturers do not act as supervisors to students? In this situation an occurrence of LECTURER will not always be related to an occurrence of STUDENT so it will be an optional relationship. However, if you consider the relationship from the STUDENT perspective it is still present as all students must have a supervising LECTURER.

To denote that a relationship can be optional a small circle is included on the relationship line at the end that is optional. The following shows the optional 1:1 relationship between STUDENT and LECTURER



There are various types of relationships

1. Complex Relationship
2. Recursive relationships

So far the concentration is on identifying and modelling relationships between pairs of entity types. Most of these relationships will be one-to-many, a few might be many-to-many and some might be one-to-one. In addition, how to resolve many-to-many relationships that contain data which is of interest in the situation being modelled has also been discovered.

Some entities are related to themselves. To be more specific, occurrences of the entity type are related to other occurrences of the same entity type. This is called a recursive relationship.

Consider the entity type EMPLOYEE in a university where there are approximately 500 employees, resulting in 500 occurrences of the entity. The Vice-chancellor manages the Deans of Faculty and each Dean manages several Heads of Department. The Heads of Department manage the lecturers. This gives rise to a hierarchical relationship within this single EMPLOYEE entity type. This can be represented graphically using a hierarchy diagram, as follows:

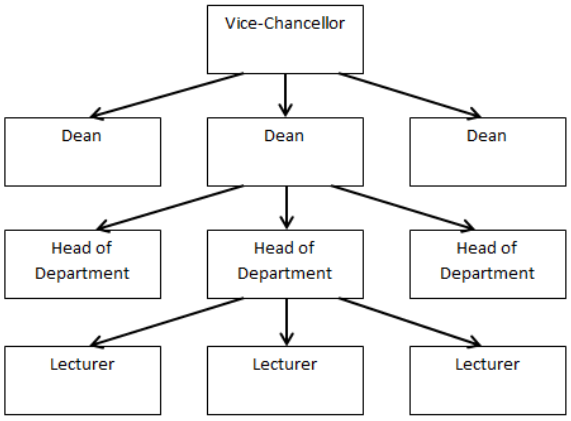
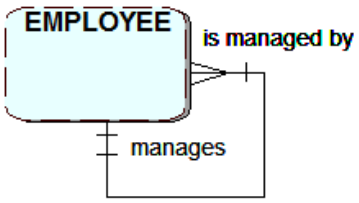


Figure 2.5: Illustration of hierarchical relationship of entity 'EMPLOYEE'

This hierarchy diagram clearly shows that an occurrence of the entity EMPLOYEE, say Dean, manages one or more other occurrences of EMPLOYEE. Another occurrence of the entity, a Head of department, also manages one or more other

occurrences of the same entity, Lecturer. To show this 1:M recursive relationship on an ERD you draw a relationship line starting and finishing at the entity, as follows:



Exclusive relationships

Sometimes two or more relationships are mutually exclusive, e.g. a VEHICLE may be undergoing a SERVICE or an INSPECTION but not both at the same time. This is shown by an arc symbol pointing towards the mutually exclusive options.

1.1.8 Logical Design Phase

In the **logical-design** phase, the designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used. The implementation data model is typically the relational data model, and this step typically consists of mapping the conceptual schema defined using the entity-relationship model into a relation schema.

It includes the following activities:

Identification of keys

Keys play a vital role in database design and have to be identified and used correctly. The following terminology is used in association with relational database keys:

-- a key uniquely identifies an entity occurrence: it is the entity identifier

-- a primary key is the key ‘chosen’ for a given relation / table

-- a candidate key is a ‘possible’ primary key (several candidate keys may exist for a relation)

-- a compound key is a key consisting of two or more attributes

1. Identification of relations

From your conceptual data model you need eventually to generate a set of relations (tables) that will form the basis of the database. You will need to link these tables in order to be able to reflect the relationships that were modelled on the ERD at the conceptual stage. In order to get to the stage of producing the tables, you first need to produce a complete set of relations in which all of the keys have been identified

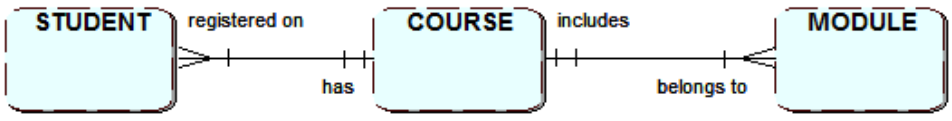
1. ***Normalize* the** database (Resolve relationships issues)
2. -- **Un**-normalised (UNF),
3. -- 1st Normal Form (1NF),
4. -- 2nd Normal Form (2NF),
5. -- 3rd Normal Form (3NF).

1.3.8 Modelling problems

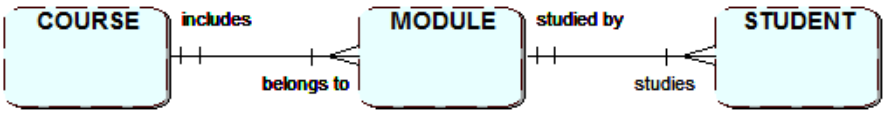
Care needs to be taken when modelling systems to ensure that you avoid producing a design which, if implemented, will not allow the system to extract all of the required information needed to answer user queries. With this is mind you should check your model for the following problems:

Fan trap

A fan trap is caused when it is not possible to link from one entity to another entity via a linking entity because the two 1:M relationships point away (fan out) from the linking entity. For example, suppose you want to know if a student is studying the module Database. The model below will not allow you to answer this query. Although the module is related to a course by the foreign key CourseID, there is no suitable link from course to student as there is no foreign key StudentID in course for student.

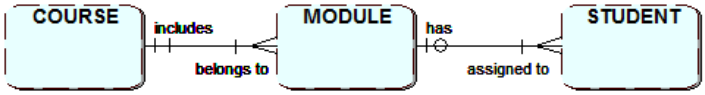


To resolve this problem a new relationship could be added, linking module directly with student, though a neater solution would be to rearrange the model as follows:-

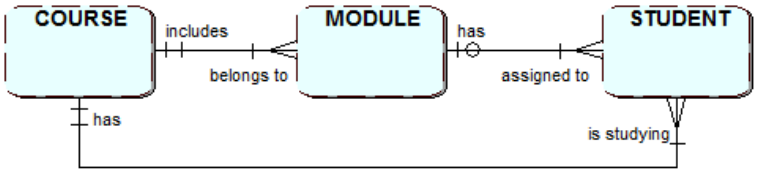


Chasm trap

A chasm trap is created when relationships between entity types indicate a route linking them, but due to optionality it is not possible to make the required connection for all occurrences. For example, suppose you want to identify which course a student is taking. The following model will not work if the student has not been assigned to a module. Although there is a link between course and module due to the Course_ID foreign key in the module relation, there would be no link between student and module if the student was not taking a module – there would be no foreign key Module_ID in student to provide a link to the module relation.



This problem can be resolved by adding a 1:M relationship linking course to student directly.



Finally, the designer uses the resulting system-specific database schema in the subsequent physical-design phase, in which the physical features of the database are specified. The physical schema of a database can be changed relatively easily after an application has been built. However, changes to the logical schema are usually harder to carry out, since they may affect a number of queries and updates scattered across application code. It is therefore important to carry out the database design phase with care, before building the rest of the database application.

The implementation phase is where you install the DBMS on the required hardware, optimize the database to run best on that hardware and software platform, and create the database and load the data. The initial data could be either new data captured directly or existing data imported from a MariaDB database or another DBMS. You also establish database security in this phase and give the various users that you've identified access applicable to their requirements. Finally, you also initiate backup plans in this phase.

The following are steps in the implementation phase:

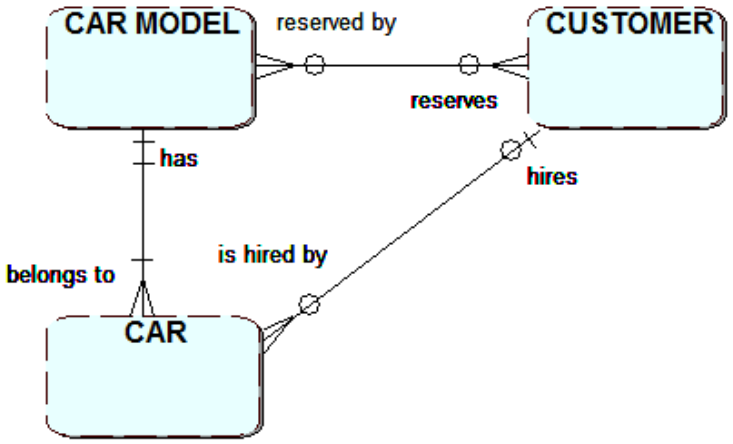
1. Install the DBMS.
2. Tune the setup variables according to the hardware, software and usage conditions.
3. Create the database and tables.
4. Load the data.
1. Set up the users and security.
2. Implement the backup regime.

1.1.8 Conclusion

A database management system provides us with mechanisms for storing and organizing data to facilitate easy access to and manipulation of the data. Today's most popular database systems are relational databases, although enhanced models exist.

1.1.9 Tutor-Marked assignment

1. Produce a logical design for the ERD



3. State reasons why normalization is needed in a database
4. Describe with suitable illustration, the various forms of normalization
5. Describe the various stages of database design

1.1.10. References and Further Readings

Atzeni, P., Ceri, S., Paraboschi, S., & Torlone, R. (1999). *Database systems: concepts, languages & architectures* (Vol. 1). London: McGraw-Hill.

Batini., C., Ceri, S., and Navathe, S. (1992) Database Design: An Entity – Relationship Approach,

Bernstein, P. (1976), Synthesizing Third Normal Form Relations from Functional Dependencies",TODS, 1:4, December 1976.

Codd, E (1970) "A Relational Model for Large Shared Data BANks" CACM, 136, June 1970.

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey. Prentice Hall

Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.

Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman

Graeme C. Simsion, Graham C. Witt (2004). Data Modeling Essentials. San Francisco. Morgan Kaufmann

Maier, D. (1983). The Theory of Relational Databases

Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

Singh, S. K. (2011). *Database systems: Concepts, design and applications*. Pearson Education India.

MODULE 1: DATABASE DESIGN AND IMPLEMENTATION

UNIT 2: ADVANCED SQL

2.1 Introduction:

Structured Query Language (SQL) is the most widely used commercial relational database language. It was designed for managing data in Relational Database Management System (RDBMS). It was originally developed at IBM in the SEQUEL XRM and System-R projects (1974-1977). Almost immediately, other vendors introduced DBMS products based on SQL. SQL continues to evolve in response to changing needs in the database area. This unit explains how to use SQL to access and manipulate data from database systems like MySQL, SQL Server, MS Access, Oracle, Sybase, DB2, and others

2.1. **OBJECTIVES**

At the end of this unit, students should be able to:

- understand the SQL statements
- be able to query a database using SQL queries

2.2 Basics Concepts of SQL

SQL - Structured Query Language is a standard language for accessing and manipulating databases. SQL lets you access and manipulate databases. It is used for defining tables and integrity constraints and for accessing and manipulating data. SQL. This unit explains how to use SQL to access and manipulate data from database systems like MySQL, SQL Server, MS Access, Oracle, Sybase, DB2, and others. Application programs may allow users to access a database without directly using SQL, but these applications themselves must use SQL to access the database.

Although SQL is an ANSI (American National Standards Institute) standard, there are many different versions of the SQL language. However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT,

UPDATE, DELETE, INSERT, WHERE) in a similar manner. Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard.

The SQL language has several aspects to it.

The Data Manipulation Language (DML): This subset of SQL allows users to pose queries and to insert, delete, and modify rows. Queries are the main focus of this unit. We covered DML commands to insert, delete, and modify rows

The Data Definition Language (DDL): This subset of SQL supports the creation, deletion, and modification of definitions for tables and views. Integrity constraints can be defined on tables, either when the table is created or later. Although the standard does not discuss indexes, commercial implementations also provide commands for creating and deleting indexes.

Triggers and Advanced Integrity Constraints: The new SQL:1999 standard includes support for triggers, which are actions executed by the DBMS whenever changes to the database meet conditions specified in the trigger.

Embedded and Dynamic SQL: Embedded SQL features allow SQL code to be called from a host language such as C or COBOL. Dynamic SQL features allow a query to be constructed (and executed) at run-time.

Client-Server Execution and Remote Database Access: These commands control how a client application program can connect to an SQL database server, or access data from a database over a network.

Transaction Management: Various commands allow a user to explicitly control aspects of how a transaction is to be executed.

Security: SQL provides mechanisms to control users' access to data objects such as tables and views.

Advanced features: The SQL:1999 standard includes object-oriented features, recursive queries, decision support queries, and also addresses emerging areas such as data mining, spatial data, and text and XML data management.

2.3. HISTORY OF SQL

SQL was developed by IBM Research in the mid 70's and standardized by the ANSI and later by the ISO. Most database management systems implement a majority of one of these standards and add their proprietary extensions. SQL allows the retrieval, insertion, updating, and deletion of data. A database management system also includes management and administrative functions. Most – if not all –implementations also include a command-line interface (SQL/CLI) that allows for the entry and execution of the language commands, as opposed to only providing an application programming interface (API) intended for access from a graphical user interface (GUI).

The first version of SQL was developed at IBM by Andrew Richardson, Donald C. Messerly and Raymond F. Boyce in the early 1970s. This version, initially called **SEQUEL**, was designed to manipulate and retrieve data stored in IBM's original relational database product; System R. IBM patented their version of SQL in 1985, while the SQL language was not formally standardized until 1986 by the American National Standards Institute (ANSI) as SQL-86. Subsequent versions of the SQL standard have been released by ANSI and as International Organization for Standardization (ISO) standards.

Originally designed as a declarative query and data manipulation language, variations of SQL have been created by SQL database management system (DBMS) vendors that add procedural constructs, flow-of-control statements, user-defined data types, and various other language extensions. With the release of the SQL: 1999 standard, many such extensions were formally adopted as part of the SQL language via the SQL Persistent Stored Modules (SQL/PSM) portion of the standard. SQL was adopted as a standard by ANSI in 1986 and ISO in 1987.

In a nutshell, SQL can perform the following

1. SQL can execute queries against a database

2. SQL can retrieve data from a database
3. SQL can insert records in a database
4. SQL can update records in a database
5. SQL can delete records from a database
6. SQL can create new databases
7. SQL can create new tables in a database
8. SQL can create stored procedures in a database
9. SQL can create views in a database
10. SQL can set permissions on tables, procedures, and views

2.4 The Form of a Basic SQL Query

The basic form of an SQL query is as follows:

SELECT [DISTINCT] *select-list*
FROM *from-list*
WHERE *qualification*

Every query must have a SELECT clause, which specifies columns to be retained in the result, and a FROM clause, which specifies a cross-product of tables. The optional WHERE clause specifies selection conditions on the tables mentioned in the FROM clause.

2.4.1 The Syntax of a Basic SQL Query

1. The from-list in the FROM clause is a list of table names. A table name can be followed by a range variable; a range variable is particularly useful when the same table name appears more than once in the from-list.
2. The select-list is a list of (expressions involving) column names of tables named in the from-list. Column names can be prefixed by a range variable.
3. The qualification in the WHERE clause is a Boolean combination (i.e., an expression using the logical connectives AND, OR, and NOT) of conditions of the form expression op expression, where op is one of the comparison operators {<, <=,

=, <>, >=, >}. An expression is a column name, a constant, or an (arithmetic or string) expression.

4. The *DISTINCT* keyword is optional. It indicates that the table computed as an answer to this query should not contain duplicates, that is, two copies of the same row. The default is that duplicates are not eliminated.

2.5. SQL STATEMENTS

Most of the actions you need to perform on a database are done with SQL statements. Some database systems require a semicolon at the end of each SQL statement. Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server. We are using MS Access and SQL Server 2000 and we do not have to put a semicolon after each SQL statement, but some database programs force you to use it. SQL statements can be divided into two parts:

SQL statements are basically divided into four; viz;

1. Data Manipulation Language (DML)
2. Data Definition Language (DDL)
3. Data Control Language (DCL)
4. Transaction Control

2.5.1 DATA MANIPULATION LANGUAGE (DML)

DML retrieves data from the database, enters new rows, changes existing rows, and removes unwanted rows from tables in the database, respectively. The basic DML includes the following;

1. Select statement
2. Insert statement
3. Update statement
4. Delete statement
5. Merge statement

DATA DEFINITION LANGUAGE sets up, changes and removes data structures from tables. The basic Data Definition Language includes the following;

1. Create statement
2. Alter statement
3. Drop statement
4. Rename statement
5. Truncate statement
6. Comment statement

DATA CONTROL LANGUAGE (DCL) gives or removes access rights to both a database and the structures within it. The basic DCLs are;

1. Grant Statement
2. Revoke Statement

TRANSACTION CONTROL manages the changes made by the DML statements. Changes to the data can be grouped together into logical transactions. The basic Transaction control languages are;

1. Commit
2. Rollback
3. Save point

Using the following simple rules and guidelines, you can construct valid statements that are both easy to read and easy to edit

1. SQL statements are not case sensitive, unless indicated
2. SQL statements can be entered on one or many lines
3. Keywords cannot be split across lines or abbreviated
4. Clauses are usually placed on separate lines for readability
5. Indents should be used to make code readable
6. Keywords typically are entered in uppercase; all other words, such as table names and columns are entered in lowercase

2.6 Viewing the Structure of a Table

The structure of any database table can be view by using the describe clause of the SQL statement. The general syntax of the *describe* statement is given below;

DESCRIBE *table;*

For the purpose of this course two tables called Departments and Employees in the Oracle database will be used. Thus, we need to see the structure of this tables so that we will be able to familiarize ourselves with the column used in the table. To do this, we write the query;

DESCRIBE *departments;*

NAME	NULL?	TYPE
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	CARCHAR(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

From the table above, we can infer that departments table has 4 columns and that 2 of these columns are not allowed to be null.

DESCRIBE *employees;*

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

From the table above, we can infer that employees table has 11 columns and that 5 of these columns are not allowed to be null.

2.7 SQL SELECT STATEMENTS

To extract data from the database, you need to use the SQL SELECT statement. You may need to restrict the columns that are displayed. Using a SELECT statement, you can do the following;

1. **Projection:** You can use the projection capability to choose the columns in a table that you want to return by your query. You can choose as few or as many columns of the table as you require.
2. **Selection:** You can use the selection capability in SQL to choose the rows in a table that you want to return by a query. You can use various criteria to restrict the rows that you use.
3. **Joining:** You can use the join capability to bring together data that is stored in different tables by creating a link between them.

SUMMARY OF THE FUNCTIONS OF SQL

1. SQL can execute queries against a database
2. SQL can retrieve data from a database
3. SQL can insert records in a database
4. SQL can update records in a database
5. SQL can delete records from a database
6. SQL can create new databases
7. SQL can create new tables in a database
8. SQL can create stored procedures in a database
9. SQL can create views in a database
10. SQL can set permissions on tables, procedures, and views
11. SQL can allow the construction codes manipulating database

2.7.1 Using SQL for Web Site

To build a web site that shows some data from a database, you will need the following:

1. An RDBMS database program (i.e. MS Access, SQL Server, MySQL)

2. A server-side scripting language, like PHP or ASP
3. SQL

1. HTML / CSS

Relational Database Management System (RDBMS)

RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access. The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

2.8 SQL SYNTAX

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is an example of a table called "Persons":

P_Id	LastName	FirstName	Address	City
1	Akinbode	Ola	10, Odeku Str.	Lagos
2	Okafor	Chris	23, Princewill Drive	Port Harcourt
3	Amodu	Ali	20, Dauda lane	Kaduna

The table above contains three records (one for each person) and five columns (P_Id, LastName, FirstName, Address, and City).

Format of SQL Statements

Most of the actions you need to perform on a database are done with SQL statements. The following SQL statement will select all the records in the "Persons" table:

```
SELECT * FROM Persons
```

Some database systems require a semicolon at the end of each SQL statement. Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

SQL, DML and DDL

SQL can be divided into two parts: The Data Manipulation Language (DML) and the Data Definition Language (DDL).

The query and update commands form the DML part of SQL:

SELECT - extracts data from a database **UPDATE** - updates data in a database

DELETE - deletes data from a database.

INSERT INTO - inserts new data into a database

The DDL part of SQL permits database tables to be created or deleted. It also define indexes (keys), specify links between tables, and impose constraints between tables. The most important DDL statements in SQL are:

CREATE DATABASE - creates a new database

ALTER DATABASE – modifies a database

CREATE TABLE - creates a new table

ALTER TABLE - modifies a table

DROP TABLE - deletes a table

CREATE INDEX - creates an index (search key)

DROP INDEX - deletes an index

The CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database.

SQL CREATE TABLE Syntax

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server visit www.datatyperef.com

CREATE TABLE Example

Now we want to create a table called "Persons" that contains five columns:

P_Id, LastName, FirstName, Address, and City. We use the following

CREATE TABLE statement:

```
CREATE TABLE Persons
(
P_Id int,
LastName varchar(255),
FirstName varchar(255),
    Address varchar(255),
City varchar(255)
)
```

The P_Id column is of type int and will hold a number. The LastName, FirstName, Address, and City columns are of type varchar with a maximum length of 255 characters.

The empty "Persons" table will now look like this:

P Id	LastName	FirstName	Address	City

The empty table can be filled with data with the INSERT INTO statement.

2.8 The SQL SELECT Statement

The SELECT statement is used to select data from a database.

The result is stored in a result table, called the result-set.

SQL SELECT Syntax

```
SELECT column_name(s)
FROM table_name
```

and

```
SELECT * FROM table_name
```

Note: *SQL is not case sensitive. SELECT is the same as select.*

An SQL SELECT Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Akinbode	Ola	10, Odeku Str.	Lagos
2	Okafor	Chris	23, Princewill Drive	Port Harcourt
3	Amodu	Ali	20, Dauda lane	Kaduna

Now we want to select the content of the columns named "LastName" and "FirstName" from the “Persons” table above. We use the following SELECT statement:

SELECT LastName, FirstName FROM Persons

The result-set will look like this:

LastName	**FirstName**
Akinbode	Ola
Okafor	Chris
Amodu	Ali

SELECT * Example

Now we want to select all the columns from the "Persons" table.

We use the following SELECT statement:

SELECT * FROM Persons

Tip: *The asterisk (*) is a quick way of selecting all columns!*

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Akinbode	Ola	10, Odeku Str.	Lagos
2	Okafor	Chris	23, Princewill Drive	Port Harcourt
3	Amodu	Ali	20, Dauda lane	kaduna

Navigation in a Result-set

Most database software systems allow navigation in the result-set with programming functions, like:

Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, etc.

The SQL SELECT DISTINCT Statement

In a table, some of the columns may contain duplicate values. This is not a problem; however, sometimes you will want to list only the different (distinct) values in a table.

The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax

```
SELECT DISTINCT column_name(s)
FROM table_name
```

SELECT DISTINCT Example

The "PersonsOne" table:

P_Id	LastName	FirstName	Address	City
1	Akinbode	Ola	10, Odeku Str.	Lagos
2	Okafor	Chris	23, Princewill Drive	Port Harcourt
3	Amodu	Ali	20, Dauda lane	kaduna

Now we want to select only the distinct values from the column named "City" from the table above.

We use the following SELECT statement:

```
SELECT DISTINCT City FROM Persons
```

The result-set will look like this:

City
Lagos
Kaduna

SQL *WHERE Clause*

The WHERE clause is used to filter records. The WHERE clause is used to extract only those records that fulfill a specified criterion.

SQL WHERE Syntax

SELECT column_name(s)

FROM table_name

WHERE column_name operator value

WHERE Clause Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Akinbode	Ola	10, Odeku Str.	Lagos
2	Okafor	Chris	23, Princewill Drive	Porthacourt
3	Amodu	Ali	20, Dauda lane	Kaduna

Now we want to select only the persons living in the city *"Sandnes"* from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City='Kaduna'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Amodu	Ali	20, Dauda lane	Kaduna

Quotes Around Text Fields

SQL uses single quotes around text values (most database systems will also accept double quotes). Although, numeric values should not be enclosed in quotes.

For text values:

```
SELECT * FROM Persons WHERE
FirstName='Chris'
```

This is wrong:

```
SELECT * FROM Persons WHERE FirstName=Chris
```

For numeric values:

This is correct:

```
SELECT * FROM Persons WHERE Year=1965
```

This is wrong:

```
SELECT * FROM Persons WHERE Year='1965'
```

Operators Allowed in the WHERE Clause

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range

LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

Note: In some versions of SQL the <> operator may be written as !=

SQL AND & OR Operators

The AND & OR operators are used to filter records based on more than one condition. The AND operator displays a record if both the first condition and the second condition is true while the OR operator displays a record if either the first condition or the second condition is true.

AND Operator Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Akinbode	Ola	10, Odeku Str.	Lagos
2	Okafor	Chris	23, Princewill Drive	Porthacourt
3	Amodu	Ali	20, Dauda lane	Kaduna

OR Operator Example

Now we want to select only the persons with the first name equal to "Tove" OR the first name equal to "Ola":

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE FirstName='Chris'
OR FirstName='Ali'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Okafor	Chris	23, Princewill Drive	Porthacourt
3	Amodu	Ali	20, Dauda lane	Kaduna

Combining AND & OR

You can also combine AND and OR (use parenthesis to form complex expressions).

Now we want to select only the persons with the last name equal to "Svendson" AND the first name equal to "Tove" OR to "Ola":

We use the following SELECT statement:

```
SELECT * FROM Persons WHERE
LastName='Akinbode' OR
LastName=’Okafor’
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Akinbode	Ola	10, Odeku Str.	Lagos
2	Okafor	Chris	23, Princewill Drive	Porthacourt

THE ORDER BY Keyword

ORDER BY Syntax

The ORDER BY keyword is used to sort the result-set. The ORDER BY keyword is used to sort the result-set by a specified column. The ORDER BY keyword sorts the records in ascending order by default. If you want to sort the records in a descending

order, you can use the DESC keyword. The order by clause comes last in a select statement.

Syntax of the order by clause given below;

SELECT *expr*
FROM *table*
[WHERE *condition(s)*]
[ORDER BY{*column, expr}* [ASC|DESC]

ORDER BY EXAMPLE

Sorting in Descending order

SELECT last_name, job_id, department_id, hire_date

FROM employees

ORDER BY last

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
Akinbode	SA_REP	80	21-APR-19
Amodu	SA_REP	20	21-APR-19
Buba	SA_REP	80	24-MAR-19
Ngozi	ST_CLERK	50	08-MAR-19
Okafor	SA_REP	80	23-FEB-19
Sowale	ST-CLERK	50	06-FEB-19

We also sort using multiple columns.

For example,

SELECT last_name, department_id, salary FROM

employees ORDER BY department_id, salary

DESC;

2.9 INSERT STATEMENT

INSERT INTO Syntax

The **INSERT INTO** statement is used to insert new records statement is used to insert a new row in a table.

The syntax of the insert statement is;

INSERT IBTO *table [{column, [,column…..])]*
VALUES (*value [, value….]);*

In the syntax,

table is the name of the table
column is the name of the column
v*alue* is the corresponding value for the column

INSERT STATEMENT EXAMPLE

INSERTING NEW ROWS

Example:

INSERT INTO departments (department_id, department_name,
manager_id, location_id) VALUES (170, 'Public Relations',100,1700);

INSERTING ROWS WITH NULL VALUES

Implicit method example

INSERT INTO departments (department_id, department_name)
Values (30,'Purchasing');
Explicit Method example
INSERT INTO departments
Values (100, 'Finance', NULL, NULL);

INSERTING SPECIAL VALUES

Example:

INSERT INTO employees (employee_id, first_name, last_name, email, phone_number,
hire_date, job_id,salary, commission_pct, manager_id, department_id)
Values (7, 'Adeola', 'Chalse', 'ade_char', '2348039990985', SYSDATE,
'AC_ACCOUNT', 6900, NULL, 205, 100);

2.10 UPDATE STATEMENT

UPDATE STATEMENT Syntax

The UPDATE statement is used to update existing records in a table.

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Note: *Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!*

SQL UPDATE EXAMPLES

updating rows in a table

```
UPDATE employees
SET department_ID=70
WHERE employee_ID = 113;
```

SQL DELETE Statement

SQL DELETE SYNTAX

The DELETE statement is used to delete records and rows in a table

```
DELETE FROM table_name
WHERE some_column=some_value
```

SQL DELETE EXAMPLES

```
DELETE *
FROM employees;

DELETE FROM employees
Where department_id =60;
```

DELETE ALL ROWS

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name

or

DELETE * FROM table_name
```

Note: *Once records are deleted, they are gone. Undoing this statement is not possible!*

2.11 Joining Tables

The Select statement can be used to join two tables together. It can be used to extract part of Table A and part of Table B to form Table C. For example, assuming *student* and *studentclass* are two different tables. Let us examine this instruction:-

```
Select student.SID, student.name,
studentclass.classname From student,
studentclass
Where student.SID = studentclass.SID
```

This statement shows that SID, name are columns or fields from student table and classname and SID are also columns from studentclass table.

The fields in the new table to form by this instruction are:-

SID name classname

2.12 ARITHMETIC OPERATIONS

Create expressions with number and date data by using arithmetic operators. You may need to modify the way in which data is displayed, perform calculations, or look at *what-if* scenarios. These are all possible using arithmetic expressions. An arithmetic expression can contain column names, constant numeric values and arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

USING ARITHMETIC OPERATORS

Let us first extend Table persons to Table employees by adding salary column to it and adding three more records. For subsequent illustrations in this unit, the table is also assumed to have more than 5 columns.

The example below describes a scenario in which arithmetic operators can be used.

SELECT last_name, salary, salary+300

FROM employees;

This gives

LastName	Salary	Salary+300
Akinbode	4800	5100
Okafor	17000	17300
Amodu	12000	12300
Buba	9000	9300
Ngozi	7700	8000
Sowale	24000	24300

2.13 OPERATOR PRECEDENCE

If an arithmetic expression contains more than one operator, multiplication and division are evaluated first. If operators within an expression are of the same priority, then evaluation is done from left to right. Parenthesis can be used to force the expression within parentheses to be evaluated first. Multiplication and division take priority over addition and subtraction. Example of a query that shows how operator precedence works is shown below;

SELECT *last_name, salary, 12*salary+100*

FROM employees

LastName	**Salary**	**12*Salary+100**
Akinbode	4800	57700
Okafor	17000	2014100
Amodu	12000	144100
Buba	9000	108100
Ngozi	7700	92500
Sowale	24000	288100

A query that uses brackets to override the operator precedence is shown below;

SELECT last_name, salary, 12*(salary+100)

FROM employees

LastName	**Salary**	**12*(Salary+100)**
Akinbode	4800	58800
Okafor	17000	205200
Amodu	12000	145200
Buba	9000	109200
Ngozi	7700	93600
Sowale	24000	289200

DEFINING A NULL VALUE

A null is a value that is unavailable, unassigned, unknown, or inapplicable. If a row lacks the data value for a particular column, that value is said to be null, or to contain a null. Columns of any data type can contain nulls. However, some constraints, NOT NULL and PRIMARY KEY, prevent nulls from being used in the column.

A query that shows the null values is shown below;

SELECT *last_name, job_id, salary, commission_pct*

FROM *employees;*

LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
Akinbode	ST_MAN	4800	
Okafor	ST_CLERK	17000	
Amodu	ST_CLERK	12000	
Buba	ST_CLERK	9000	
Ngozi	ST_CLERK	7700	
Sowale	ST_CLERK	24000	

In the COMMISSION_PCT column in the EMPLOYEES table, notice that only a sales manager (form the job_id column) can earn a commission.

Null Values in Arithmetic Expressions

Arithmetic expressions containing a null value evaluate to null.

Example:

SELECT last_name, 12*salary*commission_pct

FROM employees;

LAST_NAME	12*SALARY*COMMISSION_PCT
Akinbode	
Okafor	
Amodu	
Buba	
Ngozi	
sowale	

Processing Multiple Tables

1. Join–a relational operation that causes two or more tables with a common domain to be combined into a single table or view.

2. Equi-join–a join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table.
3. Natural join–an equi-join in which one of the duplicate columns is eliminated in the result table.
4. Outer join–a join in which rows that do not have matching values in common columns are nonetheless included in the result table (as opposed to *inner* join, in which rows must have matching values in order to appear in the result table)
5. Union join–includes all columns from each table in the join, and an instance for each row of each table
6. Self join–Matching rows of a table with other rows from the same table

TIPS FOR DEVELOPING QUERIES

1. Be familiar with the data model (entities and relationships)
2. Understand the desired results
3. Know the attributes desired in result
4. Identify the entities that contain desired attributes
5. Review ERD
6. Construct a WHERE equality for each link
7. Fine tune with GROUP BY and HAVING clauses if needed
8. Consider the effect on unusual data

Query Efficiency Considerations

1. Instead of SELECT *, identify the specific attributes in the SELECT clause; this helps reduce network traffic of result set
2. Limit the number of subqueries; try to make everything done in a single query if possible
3. If data is to be used many times, make a separate query and store it as a view

Guidelines for Better Query Design

1. Understand how indexes are used in query processing
2. Keep optimizer statistics up-to-date
3. Use compatible data types for fields and literals
4. Write simple queries
5. Break complex queries into multiple simple parts
6. Don’t nest one query inside another query
7. Don’t combine a query with itself (if possible avoid self-joins)
8. Create temporary tables for groups of queries
9. Combine update operations
10. Retrieve only the data you need
11. Don’t have the DBMS sort without an index
12. Learn!
13. Consider the total query processing time for ad hoc queries

DATA DICTIONARY FACILITIES

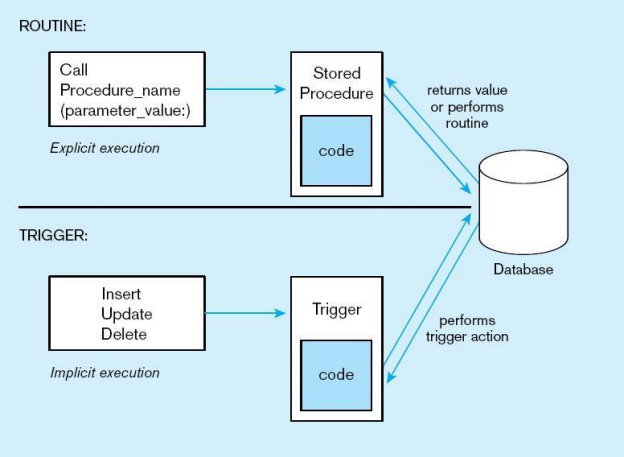
1. System tables that store metadata
2. Users usually can view some of these tables
3. Users are restricted from updating them
4. Some examples in Oracle 11g

Table	Description
DBA_TABLES	Describes all tables in the database
DBA_TAB_COMMENTS	Comments on all tables in the database
DBA_CLUSTERS	Describes all clusters in the database
DBA_TAB_COLUMNS	Describes columns of all tables, views, and clusters
DBA_COL_PRIVS	Includes all grants on columns in the database
DBA_COL_COMMENTS	Comments on all columns in tables and views
DBA_CONSTRAINTS	Constraint definitions on all tables in the database
DBA_USERS	Information about all users of the database

TRIGGERS AND ROUTINES

1. Triggers–routines that execute in response to a database event (INSERT, UPDATE, or DELETE)
2. Routines
    1. Program modules that execute on demand
3. Functions–routines that return values and take input parameters
4. Procedures–routines that do not return values and can take input or output parameters

Triggers contrasted with stored procedures



Trigger syntax in SQL

```
CREATE TRIGGER trigger_name
        {BEFORE | AFTER | INSTEAD OF} {INSERT | DELETE | UPDATE} ON
        table_name
        [FOR EACH {ROW | STATEMENT}] [WHEN (search condition)]
        <triggered SQL statement here>;
```

2.14 Some of the most important SQL Commands

1. **SELECT** - extracts data from a database
2. **UPDATE** - updates data in a database

3.	**DELETE**	-	deletes data from a database
4.	**INSERT INTO**	-	inserts new data into a database
5.	**CREATE DATABASE**	-	creates a new database
6.	**ALTER DATABASE**	-	modifies a database
7.	**CREATE TABLE**	-	creates a new table
8.	**ALTER TABLE**	-	modifies a table
9.	**DROP TABLE**	-	deletes a table
10.	**CREATE INDEX**	-	creates an index (search key)
11.	**DROP INDEX**	-	deletes an index

EMBEDDED AND DYNAMIC SQL

1. Embedded SQL

1. Including hard-coded SQL statements in a program written in another language such as C or Java

2. Dynamic SQL

1. Ability for an application program to generate SQL code on the fly, as the application is running

REASONS TO EMBED SQL IN 3GL

1. Can create a more flexible, accessible interface for the user
2. Possible performance improvement
3. Database security improvement; grant access only to the application instead of users

2.15 Conclusion

When we wish to extract information from a database, we communicate with the Database Management System (DBMS) using a query language called SQL. SQL is the most frequently used programming language in the world, in the sense that every day, more SQL programs are written, compiled and executed than programs in any other computer programming language. SQL is used with *relational* database systems. In a

relational database, all of the data is stored in tables. SQL has many built-in functions for performing calculations on data.

2.16 Tutor Marked Assignment

1. Describe the six clauses in the syntax of an SQL query, and show what type of constructs can be specified in each of the six clauses. Which of the six clauses are required and which are optional
2. What is a view in SQL, and how is it defined?

2.17 References and Further Reading:

David M. Kroenke, David J. Auer (2008). Database Concepts. New Jersey . Prentice Hall

Elmasri Navathe (2003). Fundamentals of Database Systems. England. Addison Wesley.

Fred R. McFadden, Jeffrey A. Hoffer (1994). Modern Database management. England. Addison Wesley Longman

Graeme C. Simsion, Graham C. Witt (2004). Data Modeling Essentials. San Francisco. Morgan Kaufmann

Pratt Adamski, Philip J. Pratt (2007). Concepts of Database Management. United States. Course Technology.

MODULE 1: DATABASE DESIGN AND IMPLEMENTATION

UNIT 3: DATABASE SYSTEM CATALOG

3.0 INTRODUCTION:

A relational database system needs to maintain data about the relations, such as the schema of the relations. In general, such "data about data" is referred to as metadata. Relational schemas and other metadata about relations are stored in a structure called the data dictionary or system catalog. The system catalogue is a collection of tables and views that contain important information about a database. It is the place where a relational database management system stores schema metadata, such as information about tables and columns, and internal bookkeeping information.

4.1 Objectives:

After going through this unit, you should be able to:

1. define the system catalogue and its content;
2. describe the use of catalogue in a commercial DBMS;
3. define the data dictionary system and its advantages and disadvantages, and
4. define the role of catalogue in system administration

3.2 What is a database system catalog?

As explained earlier, one of the responsibilities of DBMS is to provide a system catalog or data dictionary function regarding the various objects that are of interest to the system itself. The database system catalog is a collection of tables and views that contain important information about a database. It is the place where a relational database management system stores schema metadata, such as information about tables and columns, and internal bookkeeping information. A system catalogue is available for each database. Information in the system catalog defines the structure of the database. For example, the DDL for all tables in the database is stored in the system catalog. Most system catalogues are copied from the template database during database creation, and

are thereafter database-specific. A few catalogues are physically shared across all databases in an installation; these are marked in the descriptions of the individual catalogues.

Among the types of information that the system must store are the following:

1. Names of the relations.
2. Names of the attributes of each relation.
3. Domains and lengths of attributes.
4. Names of views defined on the database, and definitions of those views.
5. Integrity constraints (for example, key constraints).

In addition, many systems keep the following data on users of the system:

1. Names of authorized users.
2. Authorization and accounting information about users.
3. Passwords or other information used to authenticate users.

Moreso, the database may store statistical and descriptive data about the relations, such as:

• Number of tuples in each relation.

• Method of storage for each relation (for example, clustered or non-clustered).

The data dictionary may also note the storage organization (sequential, hash, or heap) of relations, and the location where each relation is stored:

1. If relations are stored in operating system files, the dictionary would note the names of the file (or files) containing each relation.
2. If the database stores all relations in a single file, the dictionary may note the blocks containing records of each relation in a data structure such as a linked list.
3. Name of the index.
4. Name of the relation being indexed.
5. Attributes on which the index is defined.
6. Type of index formed

The exact choice of how to represent system metadata by relations must be made by the system designers. One possible representation, with primary keys underlined, is shown in In this representation, the attribute index attributes of the relation Index metadata is assumed to contain a list of one or more attributes, which can be represented by a character string such as "dept name, building". The Index metadata relation is thus not in first normal form; it can be normalized, but the above representation is likely to be more efficient to access. The data dictionary is often stored in a non-normalized form to achieve fast access.

Whenever the database system needs to retrieve records from a relation, it must first consult the Relation metadata relation to find the location and storage organization of the relation, and then fetch records using this information. However, the storage organization and location of the Relation metadata relation itself

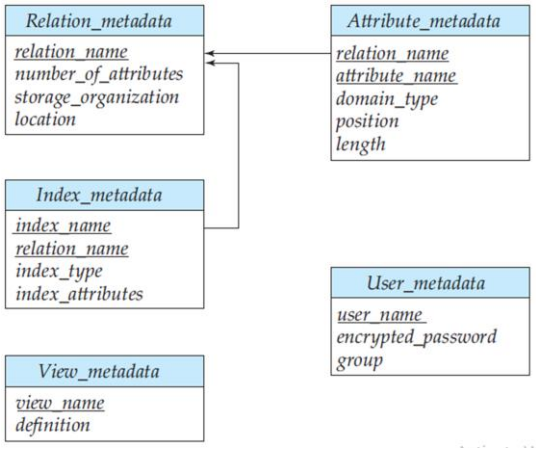


Figure 3.2: Relational schema representing system metadata

Data dictionaries also include data on the secondary keys, indexes and views. The above could also be extended to the secondary key, index as well as view information by defining the secondary key, indexes and views. Data dictionaries do not contain any actual data from the database, it contains only book-keeping information for managing it. Without a data dictionary, however, a database management system cannot access data from the database.

The Database Library is built on a Data Dictionary, which provides a complete description of record layouts and indexes of the database, for validation and efficient data access. The data dictionary can be used for automated database creation, including building tables, indexes, and referential constraints, and granting access rights to individual users and groups. The database dictionary supports the concept of Attached Objects, which allow database records to include compressed BLOBs (Binary Large Objects) containing images, texts, sounds, video, documents, spreadsheets, or programmer-defined data types.

3.3 Functions of Data Dictionary:

The data dictionary stores useful metadata, such as field descriptions, in a format that is independent of the underlying database system.

1. Ensuring efficient data access, especially with regard to the utilization of indexes,
2. Partitioning the database into both logical and physical regions,
3. Specifying validation criteria and referential constraints to be automatically enforced,
4. Supplying pre-defined record types for Rich Client features, such as security and administration facilities, attached objects, and distributed processing (i.e., grid and cluster supercomputing).

3.3.1 Features of system Catalog

A comprehensive data dictionary product will include possess the following features:

5. support for standard entity types (elements, records, files, reports, programs, systems, screens, users, terminals, etc.), and their various characteristics (e.g., for elements, the dictionary might maintain Business name, Business definition, name, Data type, Size, Format, Range(s), Validation criteria, etc.)
6. support for user-designed entity types (this is often called the "extensibility" feature); this facility is often exploited in support of data modelling, to record and cross-reference entities, relationships, data flows, data stores, processes, etc.
7. the ability to distinguish between versions of entities (e.g., test and production)

8. enforcement of in-house standards and conventions.
9. comprehensive reporting facilities, some of the reports include:
    1. detail reports of entities
    2. summary reports of entities
    3. component reports (e.g., record-element structures)
    4. cross-reference reports (e.g., element keyword indexes)
    5. where-used reports (e.g., element-record-program cross-references).
10. a query facility, both for administrators and casual users, which includes the ability to perform generic searches on business definitions, user descriptions, synonyms, etc.
11. language interfaces, to allow, for example, standard record layouts to be
12. automatically incorporated into programs during the compile process.
13. automated input facilities (e.g., to load record descriptions from a copy library).
14. security features
15. adequate performance tuning abilities
16. support for DBMS administration, such as automatic generation of DDL

Limitations of System Catalog

1. A system catalog is a useful management tool, but it also pose several challenges. It needs careful planning. We would need to define the exact requirements designing its contents, testing, implementation and evaluation.
2. The cost of a system catalog includes not only the initial price of its installation and any hardware requirements, but also the cost of collecting the information entering it into the DDS, keeping it up-to date and enforcing standards.
3. The use of a system catalog requires management commitment, which is not easy to achieve, particularly where the benefits are intangible and long term.

System Catalog in ORACLE

Meta data - data dictionary: Information about schema objects: tables, indexes, views, triggers,

Meta data are divided into three levels:

- information for objects owned by a user
- information for objects owned by a user as well as the objects that the user has been granted access to
- information about all database objects

Meta data are divided into three levels - three kinds of views:

- view name prefixed with USER
- view name prefixed with ALL
- view name prefixed with DBA

3.3.1 Views with the Prefix USER

The views most likely to be of interest to typical database users are those with the prefix USER. These views are as follows:

- refer to the user's own private environment in the database, including information about schema objects created by the user, grants made by the user, and so on,
- display only rows pertinent to the user,
- have columns identical to the other views, except that the column OWNER is implied,
- return a subset of the information in the ALL views,
- can have abbreviated PUBLIC synonyms for the sake of convenience.

  For example, the following query returns all the objects contained in a schema:

SELECT object_name, object_type FROM USER_OBJECTS;

3.3.2 Views with the Prefix ALL

Views with the prefix ALL refer to the user's overall perspective of the database. These views return information about schema objects to which the user has access through public or explicit grants of privileges and roles, in addition to schema objects that the user

owns. For example, the following query returns information on all the objects to which the user has access:

SELECT owner, object_name, object_type FROM ALL_OBJECTS;

Views with the Prefix DBA

Views with the prefix DBA show a global view of the entire database. Synonyms are not created for these views, because DBA views should be queried only by administrators. Therefore, to query DBA views, administrators must prefix the view name with its owner, SYS, as in the following:

SELECT owner, object_name, object_type

3.3.2.3 Role of System Catalog in Database Administration

Database administration is a specialized database activity that is performed by a database administrator. The system catalogue has an important role to play in the database administration. Some of the key areas where the system catalogue helps the database administrator are defined below:

1. **Enforcement of Database Integrity:** System catalogue is used to store information on keys, constraints, referential integrity, business rules, triggering events etc. on various tables and related objects. Thus, integrity enforcement would necessarily require the use of a data dictionary.
2. **Enforcement of Security:** The data dictionary also stores information on various users of the database systems and their access rights. Thus, enforcement of any security policy has to be processed through the data dictionary.
3. **Support for Database System Performance:** The data dictionary contains information on the indexes, statistics etc. Such information is very useful for query optimization. Also such information can be used by the database administrator to suggest changes in the internal schema.
4. Data dictionary can also support the process of database application development and testing as they contain the basic documentation while the systems are in the process of being developed.

3.4. Conclusion

The catalogue should normally be self-describing i.e. it should include entries describing the catalogue relvars themselves. System catalog plays a significant role in database activities and in database administration by enforcing integrity, enhancing security and supports database system performance.

3.5 Summary:

This unit provides a detailed view of a data dictionary in a DBMS. The data dictionary is one of the most important implementation tools in a database system. The system catalogue provides all the information on various entities, attributes, database statistics etc. It is a very useful tool if implemented actively in a database system. However, active implementation of a data dictionary is costly.

In this unit we have discussed concepts related to data dictionary and its use in oracle by the different types of users. We have also presented information on the data dictionary system and its advantages and disadvantages. We have provided a brief introduction to system catalogue in distributed systems and how data dictionary is useful in system administration.

3.6. Tutor-Marked Assignment (TMA)

1. What is a database catalogue system?
2. Describe the contents of a system Catlog
3. Mention the various benefits of system catalog in the administration of database
4. Discuss in detail, the features of the system catalog
5. List the disadvantages of a data dictionary

3.7 References and Further Readings

Atzeni, P., Ceri, S., Paraboschi, S., & Torlone, R. (1999). *Database systems: concepts, languages & architectures* (Vol. 1). London: McGraw-Hill.

Lorents, A. C., & Morgan, J. N. (1997). *Database Systems: Concepts, Management and Applications*. Harcourt Brace College Publishers.

Cho, H. (1997, August). Catalog management in heterogeneous distributed database systems. In *1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM. 10 Years Networking the Pacific Rim, 1987-1997* (Vol. 2, pp. 659-662). IEEE.

Kim, D., Lee, S. G., Chun, J., Park, S., Oh, J., Shillimdong, K., & San, Y. N. (2003). Catalog management in e-Commerce systems. *Proceeding of Comp. Sci. & Technology*.

MODULE 2: DBMS ADVANCE FEATURES

UNIT 1: QUERY PROCESSING & EVALUATION

2.1.0 Introduction:

The main purpose of storing data in a database is to be able to query it. A query *q* is just a mapping which takes a database instance *D* and maps it to a relation *q(D)* of fixed arity. Query processing refers to the range of activities involved in extracting data from a database. The activities include translation of queries in high-level database languages into expressions that can be used at the physical level of the file system, a variety of query-optimizing transformations, and actual evaluation of queries.

2.1.2 Objectives

At the end of this unit, students should be able to:

1. understand the basic concepts underlying the steps in query processing and optimization
2. estimate costs involved in query processing
3. apply query optimization techniques;

2.1.3 Query Processing

There are large numbers of possible strategies for processing a query, especially if the query is complex. However, it is usually worthwhile for the system to spend a substantial amount of time on the selection of a strategy. Typically, strategy selection can be done using information available in main memory, with little or no disk accesses. The actual execution of the query will involve many accesses to disk. Since the transfer of data from disk is slow relative to the speed of main memory and the central processor of the computer system, it is advantageous to spend a considerable amount of processing to save disk accesses.

2.1.3.1 Basic steps in Query Processing

Query Processing is the step by step process of breaking the high level language into low level language which machine can understand and perform the requested action for user. Query processor in the DBMS performs this task of query processing. The steps involved in processing are as follows:

1. Parsing and translation.
2. Optimization.
3. Evaluation
4. Execution

Before query processing can begin, the system must translate the query into a usable form. A language such as SQL is suitable for human use, but is ill suited to be the system's internal representation of a query. A more useful internal representation is one based on the extended relational algebra. Thus, the first action the system must take in query processing is to translate a given query into its internal form. This translation process is similar to the work performed by the parser of a compiler. In generating the internal form of the query, the parser checks the syntax of the user's query, verifies that the relation names appearing in the query are names of the relations in the database, and so on. The system constructs a parse-tree representation of the query, which it then translates into a relational-algebra expression. If the query was expressed in terms of a view, the translation phase also replaces all uses of the view by the relational-algebra expression that defines the view. The basic steps is described in figure 2.1.1

Given a query, there are generally a variety of methods for computing the answer. For example, we have seen that, in SQL, a query could be expressed in several different ways. Each SQL query can itself be translated into a relational algebra expression in one of several ways. More so, the relational-algebra representation of a query specifies only partially how to evaluate a query; there are usually several ways to evaluate relational-algebra expressions.

A relation algebra operation annotated with instructions on how to evaluate it is called an **evaluation primitive.** A sequence of primitive operations that can be used to evaluate a query is **a query-execution plan** or **query-evaluation plan**. The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

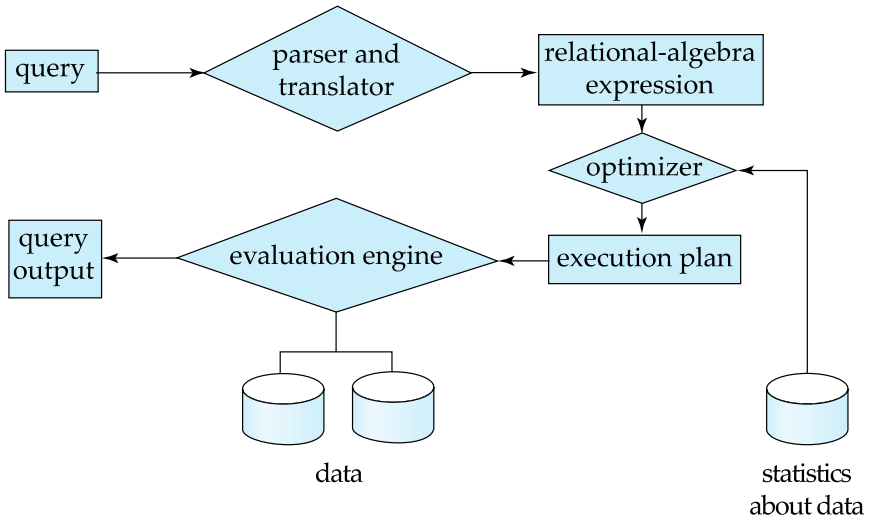


Figure 2.1.1: Stages of query processing

The different evaluation plans for a given query can have different costs. We do not expect users to write their queries in a way that suggests the most efficient evaluation plan. Rather, it is the responsibility of the system to construct a query evaluation plan that minimizes the cost of query evaluation; this task is called query optimization. Once the query plan is chosen, the query is evaluated with that plan, and the result of the query is output. The summary of the main steps are as follows:

1. **Parsing and Translating**

   Any query issued to the database is first picked by query processor. It scans and parses the query into individual tokens and examines for the correctness of query. It checks for the validity of tables / views used and the syntax of the query. Once it is passed, then it converts each tokens into relational expressions, trees and graphs. These are easily processed by the other parsers in the DBMS.

2. **Evaluation** – The query execution engine takes a physical query plan (aka execution plan), executes the plan, and returns the result.

Query evaluation is performed in the order as follows:

1. The tables in the *from clause* are combined using Cartesian products.
2. The where predicate is then applied.
3. The resulting tuples are grouped according to the group by clause.
4. The having predicate is applied to each group, possibly eliminating some groups.
5. The aggregates are applied to each remaining group. The select clause is performed last.

6. **Optimization:** Find the "cheapest" execution plan for a query

2.1.4.0 Catalog Information for Cost Estimation

Information about relations and attributes:

1. N_R: number of tuples in the relation R.
2. B_R: number of blocks that contain tuples of the relation R.
3. S_R: size of a tuple of R.
4. F_R: blocking factor; number of tuples from R that fit into one block ($F_R = [N_R/B_R]$)
5. V(A, R): number of distinct values for attribute A in R.
6. SC(A, R): selectivity of attribute A

   ≡ average number of tuples of R that satisfy an equality condition on A.

   $SC(A, R) = N_R/V(A,R)$.

Information about indexes:

1. HT_I: number of levels in index I (B^+-tree).
2. LB_I: number of blocks occupied by leaf nodes in index I (first-level blocks).
3. Val_I: number of distinct values for the search key

2.1.5.0 Measures of Query Cost

There are multiple possible evaluation plans for a query, and it is important to be able to compare the alternatives in terms of their estimated cost, and choose the best plan. To do so, we must estimate the cost of individual operations, and combine them to get the cost of a query evaluation plan. Disk accesses time, CPU time, or even communication overhead in a distributed or parallel system are some commonly used measures. Typically disk access is the predominant cost, and is also relatively easy to estimate. Therefore number of block transfers from disk is used as a measure of the actual cost of evaluation. It is assumed that all transfers of blocks have the same cost.

In large database systems, the cost to access data from disk is usually the most important cost, since disk accesses are slow compared to in-memory operations. Moreover, CPU speeds have been improving much faster than have disk speeds. Thus, it is likely that the time spent in disk activity will continue to dominate the total time to execute a query. The CPU time taken for a task is harder to estimate since it depends on low-level details of the execution code. Although real-life query optimizers do take CPU costs into account, for simplicity in this book we ignore CPU costs and use only disk-access costs to measure the cost of a query-evaluation plan.

We use the *number of block transfers* from disk and the *number of disk seeks* to estimate the cost of a query-evaluation plan. If the disk subsystem takes an average of t_T seconds to transfer a block of data, and has an average block-access time (disk seek time plus rotational latency) of t_S seconds, then an operation that transfers b blocks and performs S seeks would take $b * t_T + S * t_S$ seconds. The values of t_T and t_S must be calibrated for the disk system used, but typical values for high-end disks today would be t_S = 4 milliseconds and t_T = 0.1 milliseconds, assuming a 4-kilobyte block size and a transfer rate of 40 megabytes per second.

We can refine our cost estimates further by distinguishing block reads from block writes, since block writes are typically about twice as expensive as reads (this is because disk systems read sectors back after they are written to verify that the write was

successful). For simplicity, we ignore this detail, and leave it to you to work out more precise cost estimates for various operations. The cost estimates we give do not include the cost of writing the final result of an operation back to disk. These are taken into account separately where required.

The costs of all the algorithms that we consider depend on the size of the buffer in main memory. In the best case, all data can be read into the buffers, and the disk does not need to be accessed again. In the worst case, we assume that the buffer can hold only a few blocks of data—approximately one block per relation. When presenting cost estimates, we generally assume the worst case. In addition, although we assume that data must be read from disk initially, it is possible that a block that is accessed is already present in the in-memory buffer. Again, for simplicity, we ignore this effect; as a result, the actual disk-access cost during the execution of a plan may be less than the estimated cost. The response time for a query-evaluation plan (that is, the wall-clock time required to execute the plan), assuming no other activity is going on in the computer, would account for all these costs, and could be used as a measure of the cost of the plan. Unfortunately, the response time of a plan is very hard to estimate without actually executing the plan, for the following reasons:

1. The response time depends on the contents of the buffer when the query begins execution; this information is not available when the query is optimized, and is hard to account for even if it were available.
2. In a system with multiple disks, the response time depends on how accesses are distributed among disks, which is hard to estimate without detailed knowledge of data layout on disk.

Interestingly, a plan may get a better response time at the cost of extra resource consumption. For example, if a system has multiple disks, a plan A that requires extra disk reads, but performs the reads in parallel across multiple disks may finish faster than another plan B that has fewer disk reads, but from only one disk. However, if many instances of a query using plan A run concurrently, the overall response time may

actually be more than if the same instances are executed using plan B, since plan A generates more load on the disks. As a result, instead of trying to minimize the response time, optimizers generally try to minimize the total **resource consumption** of a query plan.

2.1.5.1 Selection Operation

In query processing, the file scan is the lowest-level operator to access data. File scans are search algorithms that locate and retrieve records that fulfill a selection condition. In relational systems, a file scan allows an entire relation to be read in those cases where the relation is stored in a single, dedicated file.

Linear search

Scan each file block and test all records to see whether they satisfy the selection condition.

– Cost estimate (number of disk blocks scanned) EA1 = br

– If selection is on a key attribute, E_{A1} = (br/ 2) (stop on finding record)

– Linear search can be applied regardless of

∗ selection condition, or

∗ ordering of records in the file, or

∗ availability of indices

– Expensive, but always applicable.

A2 (binary search). The file ordered based on attribute A (primary index). Applicable if selection is an equality comparison on the attribute on which file is ordered.

– Assume that the blocks of a relation are stored contiguously

– Cost estimate (number of disk blocks to be scanned):

$E_{A2} = [\log_2(b_r)] + [SC(A,r)/f_r] - 1$

* $[\log_2(br)]$ — cost of locating the first tuple by a binary search on the blocks

* SC(A, r) — number of records that will satisfy the selection

∗ $[SC(A, r)/f_r]$ — number of blocks that these records will occupy

— Equality condition on a key attribute: SC(A,r) = 1; estimate reduces to $E_{A2} = [\log_2(b_r)]$

- Binary search - Applicable only when the file is appropriately ordered.

1. Hash index search
2. Single record retrieval; does not work for range queries. „ Retrieval of multiple records.
3. Clustering index search - multiple records for each index item.
4. Implemented with single pointer to block with first associated record.
5. Secondary index search - Implemented with dense pointers, each to a single record
6. Index structures are referred to as access paths, since they provide a path through which data can be located and accessed.
7. A primary index is an index that allows the records of a file to be read in an order that corresponds to the physical order in the file.
8. An index that is not a primary index is called a secondary index.
9. Search algorithms that use an index are referred to as index scans.

Example (for Employee DB)

- $F_{Employee}$ = 10;
  $V_{(Deptno, Employee)}$ = 50 (different departments)
- $N_{Employee}$ = 10, 000 (Relation Employee has 10,000 tuples)
- Assume selection σ_{Deptno}=20(Employee) and Employee is sorted on search key Deptno :

=⇒ 10,000/50 = 200 tuples in Employee belong to Deptno 20; (assuming an equal distribution)

200/10 = 20 blocks for these tuples

=⇒ A binary search finding the first block would require $\log_2(1, 000)]$ = 10 block accesses

Total cost of binary search is 10+20 block accesses (versus 1,000 for linear search and Employee not sorted by Deptno)

2.1.5.2 Index scan – search algorithms that use an index (here, a B^+-tree); selection condition is on search key of index

• S3 – Primary index I for A, A primary key, equality A = a

cost(S3) = HT_I + 1 (only 1 tuple satisfies condition)

• S4 – Primary index I on non-key A equality A = a
cost(S4) = HT_I + [SC(A, R)/ F_R]

S5 – Non-primary (non-clustered) index on non-key A, equality A = a

Cost (S5) = HT_I + SC(A, R)
Worst case: each matching record resides in a different block.

• **Example (Cont.):**
– Assume primary (B^+-tree) index for attribute Deptno
– 200/10=20 blocks accesses are required to read Employee tuples
– If B^+-tree index stores 20 pointers per (inner) node, then the B^+-tree index must have between 3 and 5 leaf nodes and the entire tree has a depth of 2

=⇒ a total of 22 blocks must be read.

Selections Involving Comparisons
• Selections of the form $\sigma_{A\leq v}(R)$ or $\sigma_{A\geq v}(R)$ are implemented using a file scan or binary search, or by using either a
– **S6** – A primary index on A, or
– **S7** – A secondary index on A (in this case, typically a linear file scan may be cheaper; but this depends on the selectivity of A)

Complex Selections
• General pattern:
– Conjunction – $\sigma_{\Theta 1 \wedge \ldots \wedge \Theta n}(R)$
– Disjunction – $\sigma_{\Theta 1 \vee \ldots \vee \Theta n}(R)$
– Negation – $\sigma_{\neg\Theta}(R)$
• The selectivity of a condition Θ_i is the probability that a tuple in the relation R satisfies Θ_i . If s_i is the number of tuples in R that satisfy Θ_i , then Θ_i's selectivity is estimated as s_i/N_R.

2.1.5.4 Join Operation

The database join operation is used to combine tuples from two different relations based on some common information. For example, a course-offering relation could contain information concerning all classes offered at a university and a *student-registration* relation could contain information for which courses a student has registered. A join would typically be used to produce a student schedule, which includes data about required textbooks, time, and location of courses, as well as general student identification information. It is used to combine tuples from two or more relations. Tuples are combined when they satisfy a specified join condition.

1. Choice of a particular algorithm is based on cost estimate
2. For this, join size estimates are required and in particular cost estimates for outer-level operations in a relational algebra expression.

- **Example**: Assume the query CUSTOMERS ⋈ORDERS (with join attribute only being CName)
  - $N_{CUSTOMERS}$ = 5,000 tuples
  - $F_{CUSTOMERS}$ = 20, that is., $B_{CUSTOMERS}$ = 5,000/20 = 250 blocks
  - N_{ORDERS} = 10,000 tuples
  - F_{ORDERS} = 25, i.e., BORDERS = 400 blocks
  - V(CName, ORDERS) = 2,500, meaning that in this relation, on average, each customer has four orders
  - Also assume that CName in ORDERS is a foreign key on CUSTOMERS

Join Algorithms

1. Nested loop join
2. Index-based join
3. Sort-merge join
4. Hash join

5. Nested-Loop Join: (nested-loop-join)

6. This algorithm is called the **nested-loop join** algorithm, since it basically consists of a pair of nested **for** loops. Relation *r* is called the **outer relation** and relation *s* the **inner relation** of the join, since the loop for *r* encloses the loop for *s*. The algorithm uses the notation *tr*·*ts*, where *tr* and *ts* are tuples; *tr*·*ts* denotes the tuple constructed by concatenating the attribute values of tuples t_r and t_s.

```
For each r in R do
        For each s in S do
                if r.C = s.C then output r,s pair
        end
end
```

- The simplest algorithm. It works, but may not be efficient.

1. Exhaustive comparison (i.e., brute force approach)
2. The ordering (outer/inner) of files and allocation of buffer space is important.
3.
4. **Index Join:**

```
For each r in R do
        X <- index-lookup (S.C, r.C)
        For each s in X do
        output (r,s)
```

Look up index to find matching tuples from S.

5. If an index is available on the inner loop's join attribute and join is an equi-join or natural join, more efficient index lookups can replace file scans.
6. It is even possible (reasonable) to construct index just to compute a join.
7. For each tuple t_R in the outer relation R, use the index to lookup tuples in S that satisfy join condition with t_R
8. Worst case: db buffer has space for only one page of R and one page of the index associated with S:

   BR disk accesses to read R, and for each tuple in R, perform index lookup on S.

– Cost of the join: BR + NR ∗ c, where c is the cost of a single selection on S using the join condition.

9. If indexes are available on both R and S, use the one with the fewer tuples as the outer relation.

Example:

– Compute CUSTOMERS ⋈ ORDERS, with CUSTOMERS as the outer relation.

– Let ORDERS have a primary B^+-tree index on the join attribute CName, which contains 20 entries per index node

– Since ORDERS has 10,000 tuples, the height of the tree is 4, and one more access is needed to find the actual data records (based on tuple identifier).

– Since $N_{CUSTOMERS}$ is 5,000, the total cost is $250 + 5000 * 5 = 25{,}250$ disk accesses.

– This cost is lower than the 100,250 accesses needed for a block nested-loop join.

10. **Sort-Merge Join:** If tables have been sorted by the join attribute, we need to scan each table only once.

– Maintain one cursor per table and move the cursor forward. • Sort tables and join them.

11. first sort both relations on join attribute (if not already sorted this way)
12. Join steps are similar to the merge stage in the external sort-merge algorithm
13. Every pair with same value on join attribute must be matched

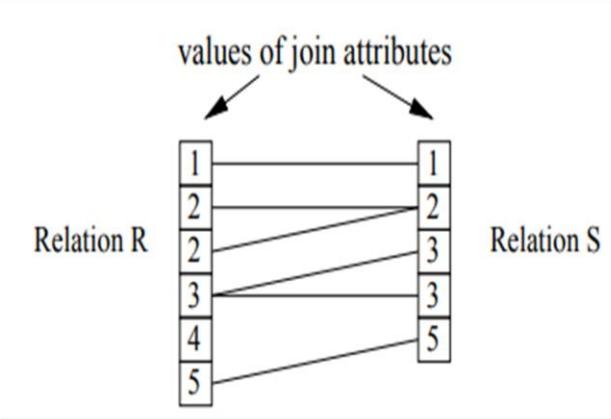


Figure 2.1.2: Query join Operation

1. If no repeated join attribute values, each tuple needs to be read only once. As a result, each block is read only once. Thus, the number of block accesses is $B_R + B_S$ (plus the cost of sorting, if relations are unsorted).
2. Worst case: all join attribute values are the same. Then the number of block accesses is $B_R + B_R * B_S$.
3. If one relation is sorted and the other has a secondary B^+-tree index on the join attribute, a **hybrid merge-join** is possible. The sorted relation is merged with the leaf node entries of the B+-tree. The result is sorted on the addresses (rids) of the

unsorted relation's tuples, and then the addresses can be replaced by the actual tuples efficiently

Hash Join:

– only applicable in case of equi-join or natural join

– a hash function is used to partition tuples of both relations into sets that have the same hash value on the join attribute

Partitioning Phase: $2 * (B_R + B_S)$ block accesses Matching Phase: $B_R + B_S$ block accesses (under the assumption that one partition of each relation fits into the database buffer)

Cost Estimates for other Operations

Sorting:

1. If whole relation fits into db buffer ➔ quick-sort
2. Or, build index on the relation, and use index to read relation in sorted order.
3. Relation that does not fit into db buffer ➔ external sort-merge
    1. **Phase:** Create runs by sorting portions of the relation in db buffer
    2. **Phase:** Read runs from disk and merge runs in sort order

Duplicate Elimination:

• **Sorting:** remove all but one copy of tuples having identical value(s) on projection attribute(s)

• **Hashing:** partition relation using hash function on projection attribute(s); then read partitions into buffer and create in-memory hash index; tuple is only inserted into index if not already present Set Operations:

• **Sorting or hashing**

• **Hashing:** Partition both relations using the same hash function; use in-memory index for partitions R_i

$R \cup S$: if tuple in R_i or in S_i, add tuple to result $\cap$: if tuple in R_i and in S_i, . . .
$-$: if tuple in R_i and not in S_i, . . .

Grouping and aggregation:

• Compute groups via sorting or hashing.

• **Hashing:** while groups (partitions) are built, compute partial aggregate values (for group attribute A, V(A,R) tuples to store values)

Evaluation of Expressions

Now let us try to understand how DBMS evaluates the query written in SQL. i.e.; how it breaks them into pieces to get the records quickly.

• **Strategy 1:** materialization. Evaluate one operation at a time, starting at the lowest level. Use intermediate results materialized in temporary relations to evaluate next level operation(s).

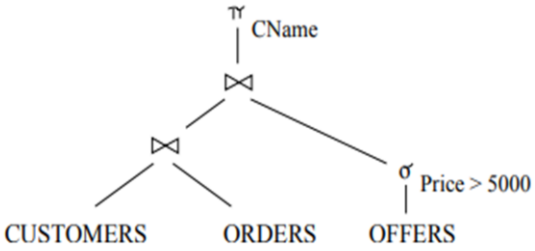


Figure 2.1.3: Illustration of materialization

1. First compute and store $\sigma_{Price}>5000$(OFFERS); then compute and store join of CUSTOMERS and ORDERS; finally, join the two materialized relations and project on to CName

2. **Strategy 2: Pipelining.**

   In this method, DBMS do not store the records into temporary tables. Instead, it queries each query and result of which will be passed to next query to process and so on. It will process the query one after the other and each will use the result of previous query for its processing. That is, DBMS evaluate several operations simultaneously, and pass the result (tuple- or block-wise) on to the next operation.

In the example above, once a tuple from OFFERS satisfying selection condition has been found, pass it on to the join. Similarly, do not store result of (final) join, but pass tuples directly to projection. • Much cheaper than materialization, because temporary relations are not generated and stored on disk.

1. **Pipelining** is not always possible, e.g., for all operations that include sorting (blocking operation).
2. **Pipelining** can be executed in two ways, demand driven and producer driven fashion

3. demand driven (lazy evaluation)

In this method, the result of lower level queries are not passed to the higher level automatically. It will be passed to higher level only when it is requested by the higher level. In this method, it retains the result value and state with it and it will be transferred to the next level only when it is requested.

1. Producer driven fashion.

In this method, the lower level queries eagerly pass the results to higher level queries. It does not wait for the higher level queries to request for the results. In this method, lower level query creates a buffer to store the results and the higher level queries pulls the results for its use. If the buffer is full, then the lower level query waits for the higher level query to empty it. Hence it is also called as PULL and PUSH pipelining.

There are still more methods of pipelining like Linear and non-linear methods of pipelining, left deep tree, right deep tree etc.

Transformation of Relational Expressions

Generating a query-evaluation plan for an expression of the relational algebra involves two steps:

1. Generate logically equivalent expressions
2. Annotate these evaluation plans by specific algorithms and access structures to get alternative query plans

1. Use equivalence rules to transform a relational algebra expression into an equivalent one.
2. Based on estimated cost, the most cost-effective annotated plan is selected for evaluation. The process is called cost-based query optimizatio

Transformation of Relational Expressions

When a SQL query is submitted to DB, it can be evaluated in number of ways. For example, consider the below case:

```
SELECT EMP_ID, DEPT_NAME
FROM EMP, DEPT
WHERE EMP.DEPT_ID = DEPT.DEPT_ID
AND EMP.DEPT_ID = 10;
```

Above query selects the EMP_ID and DEPT_NAME from EMP and DEPT table for DEPT_ID = 10. But when it is given to the DBMS, it divides the query into tokens and sees how it can be put together so that performance will be better. This is the duty of query optimizer. But altering the order of tokens in the query should not change the result. In either way it should give same result. Order of records can change and are least important. This is called equivalent query. There is set of rules to put tokens in the query. This is called **equivalence rule**

1. Select the records of EMP with DEPT_ID = 10 first then join them with DEPT table to get all matching records of EMP and DEPT. Then select only the columns EMP_ID and DEPT_NAME to display the result.
2. Select all matching records from EMP and DEPT, from which filter on DEPT_ID = 10 and select only EMP_ID and DEPT_NAME to display.

Both the steps above are same irrespective of how it is performed. Hence both are called equivalent query. These are not written in SQL, but using relational algebra, graph or tree.

$\prod$ EMP_ID, DEPT_NAME (σ DEPT_ID = 10 (EMP ∞DEPT))

or

σ DEPT_ID = 10 ($\prod$ EMP_ID, DEPT_NAME, DEPT_ID (EMP ∞DEPT))

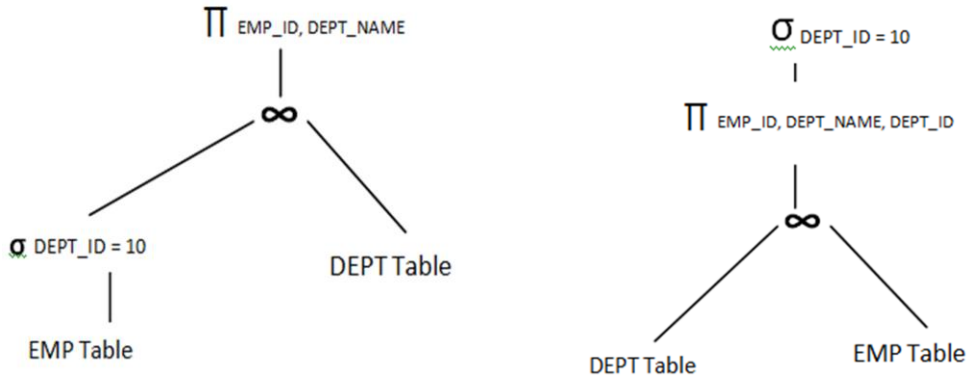


Figure 2.1.4: Example of relational Algebra tree

Above relational algebra and tree shows how DBMS depicts the query inside it. But the cost of both of them may vary. This is because the number of records in each step changes depending on the join and filters we use, and the algorithms used to evaluate them. For example we may have huge number of records in second case tree above to filter. But in the first case we are filtering the record first; hence number of records to join would have been reduced. This makes lots of difference and query optimizer calculates this difference and selects the optimal tree for query evaluation.

Equivalence Rules (for expressions E, E_1, E_2, conditions F_i)

Applying distribution and commutativity of relational algebra operations

1. $\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$
2. $\sigma_F(E_1\ [\cup, \cap, -]\ E_2) \equiv \sigma_F(E_1)\ [\cup, \cap, -]\ \sigma_F(E_2)$
3. $\sigma_F(E_1 \times E_2) \equiv \sigma_{F0}(\sigma_{F1}(E_1) \times \sigma_{F2}(E_2))$;
   $F \equiv F0 \wedge F1 \wedge F2$, Fi contains only attributes of E_1, $i = 1, 2$.
4. $\sigma_{A=B}(E_1 \times E_2) \equiv E_1 \underset{A=B}{\bowtie} E_2$
5. $\pi_A(E_1\ [\cup, \cap, -]\ E_2) \not\equiv \pi_A(E_1)\ [\cup, \cap, -]\ \pi_A(E_2)$
6. $\pi_A(E_1 \times E_2) \equiv \pi_{A1}(E_1) \times \pi_{A2}(E_2)$,
   with $Ai = A \cap \{$ attributes in $E_1\}$, $i = 1, 2$.
7. $E_1\ [\cup, \cap]\ E_2 \equiv E_2\ [\cup, \cap]\ E_1$
   $(E_1 \cup E_2) \cup E_3 \equiv E_1 \cup (E_2 \cup E_3)$ (the analogous holds for $\cap$)
8. $E_1 \times E_2 \equiv \pi_{A1,A2}(E_2 \times E_1)$
   $(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$
   $(E_1 \times E_2) \times E_3 \equiv \pi((E_1 \times E_3) \times E_2)$
9. $E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$ $(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$

Examples:

• **Selection:**

– Find the name of all customers who have ordered a product for more than $5,000 from a supplier located in Davis.

πCName($\sigma_{\text{SAddress like 0\%Davis\%'}} \wedge$ Price>5000 (CUSTOMERS $\bowtie$ (ORDERS $\bowtie$ (OFFERS $\bowtie$ SUPPLIERS)))) Perform selection as early as possible (but take existing indexes on relations into account)

πCName(CUSTOMERS $\bowtie$ (ORDERS $\bowtie$ (σ_{Price}>5000(OFFERS) $\bowtie$ ($\sigma_{\text{SAddress like 0\%Davis\%'}}$(SUPPLIERS)))))

Projection:

– $\pi_{\text{CName,account}}$(CUSTOMERS $\bowtie$ $\sigma_{\text{Prodname}=\text{'CD–ROM'}}$(ORDERS))

Reduce the size of argument relation in join

$\pi_{\text{CName,account}}$(CUSTOMERS $\bowtie$ π_{CName}($\sigma_{\text{Prodname='CD–ROM'}}$(ORDERS)))

Projection should not be shifted before selections, because minimizing the number of tuples in general leads to more efficient plans than reducing the size of tuples

Join Ordering

- For relations R_1, R_2, R_3, $(R_1 \bowtie R_2) \bowtie R_3 \equiv R_1 \bowtie (R_2 \bowtie R_3)$
- If $(R_2 \bowtie R_3)$ is quite large and $(R_1 \bowtie R_2)$ is small, we choose $(R_1 \bowtie R_2) \bowtie R_3$ so that a smaller temporary relation is computed and materialized
- Example: List the name of all customers who have ordered a product from a supplier located in Davis.

  $\pi_{CName}(\sigma_{SAddress\ like\ 0\%Davis\%}$'(SUPPLIERS $\bowtie$ ORDERS $\bowtie$ CUSTOMERS))

ORDERS $\bowtie$CUSTOMERS is likely to be a large relation. Because it is likely that only a small fraction of suppliers are from Davis, we compute the join

$\sigma_{SAddress\ like0\%Davis\%0}$(SUPPLIERS $\bowtie$ ORDERS) first.

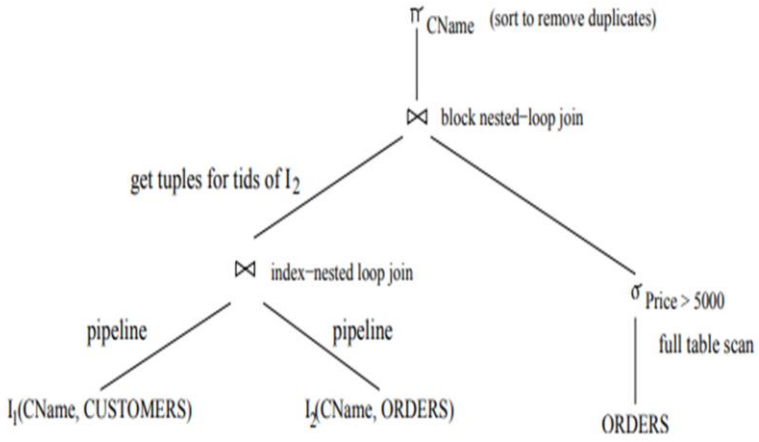
Summary of Algebraic Optimization Rules

1. Perform selection as early as possible
2. Replace Cartesian product by join whenever possible
3. Project out useless attributes early.
4. If there are several joins, perform most restrictive join first

Evaluation Plan : An evaluation plan for a query exactly defines what algorithm is used for each operation, which access structures are used (tables, indexes, clusters), and how the execution of the operations is coordinated. Example of Annotated Evaluation Plan

• Query: List the name of all customers who have ordered a product that costs more than $5,000.

Assume that for both CUSTOMERS and ORDERS an index on C_{Name} exists: I_1(CName, CUSTOMERS), I_2(CName, ORDERS).



Choice of an Evaluation Plan

- Must consider interaction of evaluation techniques when choosing evaluation plan: choosing the algorithm with the least cost for each operation independently may not yield the best overall algorithm.
- Practical query optimizers incorporate elements of the following two optimization approaches:

– Cost-based: enumerate all the plans and choose the best plan in a cost-based fashion.

– Rule-based: Use rules (heuristics) to choose plan.

• Remarks on cost-based optimization:

– Finding a join order for R1 ⋈ R2 ⋈ . . . ⋈ Rn: n! different left-deep join orders

∗ For example, for n = 9, the number is 362880. ⋈ use of dynamic programming techniques

• Heuristic (or rule-based) optimization transforms a given query tree by using a set of rules that typically (but not in all cases) improve execution performance:

– Perform selection early (reduces number of tuples)

– Perform projection early (reduces number of attributes)

– Perform most restrictive selection and join operations before other similar operations.

2.1.5.5 Query Optimization

We have seen so far how a query can be processed based on indexes and joins, and how they can be transformed into relational expressions. The query optimizer uses these two techniques to determine which process or expression to consider for evaluating the query.

A **cost-based optimizer** explores the space of all query-evaluation plans that are equivalent to the given query, and chooses the one with the least estimated cost. We have seen how equivalence rules can be used to generate equivalent plans. However, cost-based optimization with arbitrary equivalence rules is fairly complicated. We first cover a simpler version of cost-based optimization, which involves only join-order and join algorithm selection,

Suppose, we have series of table joined in a query.

$$T_1 \infty T_2 \infty T_3 \infty T_4 \infty T_5 \infty T_6$$

For above query we can have any order of evaluation. We can start taking any two tables in any order and start evaluating the query. Ideally, we can have join combinations in (2(n-1))! / (n-1)! ways. For example, suppose we have 5 tables involved in join, then we can have 8! / 4! = 1680 combinations. But when query optimizer runs, it does not evaluate in all these ways always. It uses dynamic programming where it generates the costs for join orders of any combination of tables. It is calculated and generated only once. This least cost for all the table combination is then stored in the database and is used for future use. That is.; say we have a set of tables, T ={ T_1 , T_2 , T_3 .. T_n}, then it generates least cost combination for all the tables and stores it.

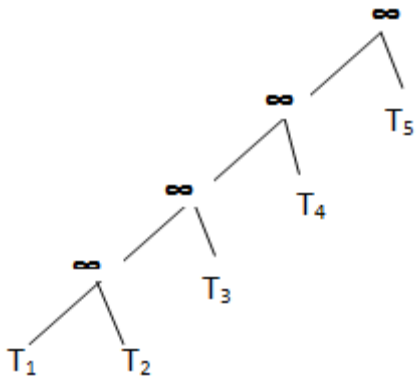
2.1.5.6 Dynamic Programming

As we learnt above, the least cost for the joins of any combination of table is generated here. These values are stored in the database and when those tables are used in the query, this combination is selected for evaluating the query. Suppose we have set of

tables, T = {T1 , T2 , T3 .. Tn}, in a DB. It picks the first table, and Computes cost for joining with rest of the tables in set T. It calculates cost for each of the tables and then chooses the best cost. It continues doing the same with rest of the tables in set T. It will generate 2n – 1 cases and it selects the lowest cost and stores it. When a query uses those tables, it checks for the costs here and that combination is used to evaluate the query. This is called dynamic programming.

1. Left Deep Trees

This is another method of determining the cost of the joins. Here, the tables and joins are represented in the form of trees. The joins always form the root of the tree and table is kept at the right side of the root. LHS of the root always point to the next join. Hence it gets deeper and deeper on LHS. Hence it is called as left deep tree. In this method, time required to find optimized query is in the order of 3n, where n is the number of tables. Suppose we have 5 tables, then time required in 35 = 243, which is lesser than finding all the combination of tables and then deciding the best combination (1680). Also, the space required for computing and storing the cost is also less and is in the order of 2n. In above example, it is 25 = 32.



Here instead of calculating the best join cost for set of tables, best join cost for joining with each table is calculated. In this method, time required to find optimized query is in the order of n2n, where n is the number of tables. Suppose we have 5 tables, then time required in 5*25 =160, which is lesser than dynamic programming. Also, the space

required for computing storing the cost is also less and is in the order of 2n. In above example, it is 25 = 32, same as dynamic programming.

1. Interesting Sort Orders

This method is an enhancement to dynamic programming. Here, while calculating the best join order costs, it also considers the sorted tables. It assumes, calculating the join orders on sorted tables would be efficient. i.e.; suppose we have unsorted tables T_1 , T_2 , T_3 .. T_n and we have join on these tables.(

$$(T_1 \infty T_2) \infty T_3 \infty \ldots \infty T_n$$

This method uses hash join or merge join method to calculate the cost. Hash Join will simply join the tables. We get sorted output in merge join method, but it is costlier than hash join. Even though merge join is costlier at this stage, when it moves to join with third table, the join will have less effort to sort the tables. This is because first table is the sorted result of first two tables. Hence it will reduce the total cost of the query.

But the number of tables involved in the join would be relatively less and this cost/space difference will be hardly noticeable. All these cost based optimizations are expensive and are suitable for large number of data. There is another method of optimization called heuristic optimization, which is better compared to cost based optimization.

2. Heuristic Optimization (Logical)

This method is also known as rule based optimization. This is based on the equivalence rule on relational expressions; hence the number of combination of queries get reduces here. Hence the cost of the query too reduces. This method creates relational tree for the given query based on the equivalence rules. These equivalence rules by providing an alternative way of writing and evaluating the query, gives the better path to evaluate the query. This rule need not be true in all cases. It needs to be examined after applying those rules. The most important set of rules followed in this method is listed below:

1. Perform all the selection operation as early as possible in the query. This should be first and foremost set of actions on the tables in the query. By performing the selection operation, we can reduce the number of records involved in the query, rather than using the whole tables throughout the query.

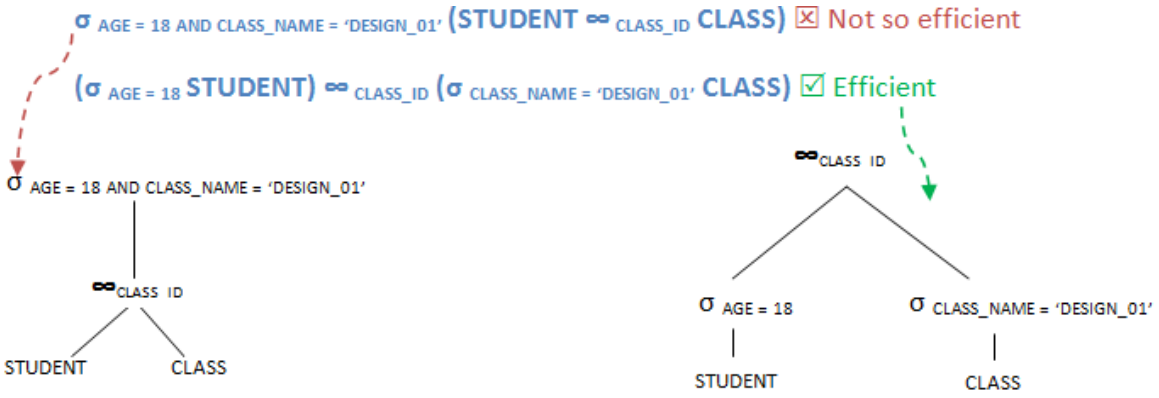
2. Suppose we have a query to retrieve the students with age 18 and studying in class DESIGN_01. We can get all the student details from STUDENT table, and class details from CLASS table. We can write this query in two different ways.

```
SELECT std.* FROM STUDENT std, CLASS cls
WHERE std.CLASS_ID = cls.CLASS_ID
AND std.AGE = 18
AND cls.CLASS_NAME = 'DESIGN_01';
```

```
SELECT std.* FROM
   (SELECT * FROM STUDENT WHERE AGE = 18) std,
   (SELECT * FROM CLASS WHERE CLASS_NAME = 'DESIGN_01') cls
WHERE std.CLASS_ID = cls.CLASS_ID
```

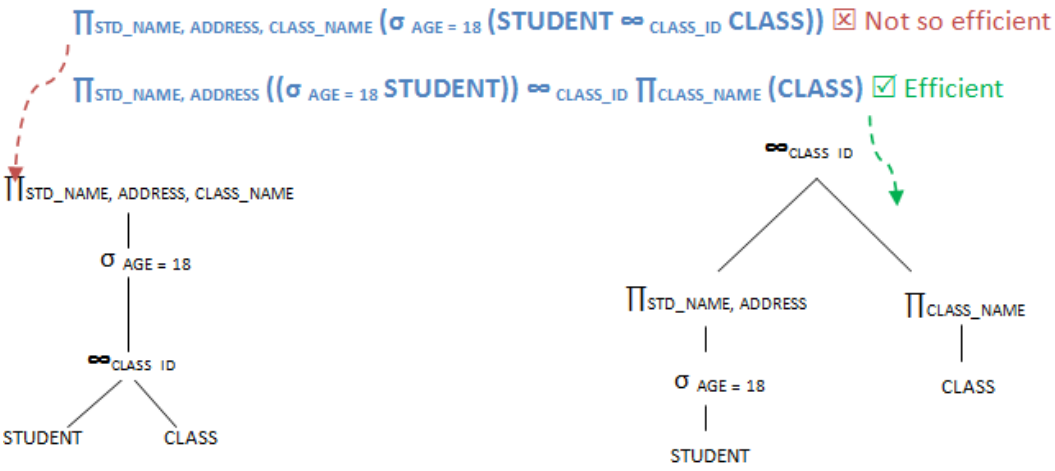
Here both the queries will return same result. But when we observe them closely we can see that first query will join the two tables first and then applies the filters. That means, it traverses whole table to join, hence the number of records involved is more. But he second query, applies the filters on each table first. This reduces the number of records on each table (in class table, the number of record reduces to one in this case!). Then it joins these intermediary tables. Hence the cost in this case is comparatively less.

Instead of writing query the optimizer creates relational algebra and tree for above case.



1. Perform all the projection as early as possible in the query. This is similar to selection but will reduce the number of columns in the query.

Suppose for example, we have to select only student name, address and class name of students with age 18 from STUDENT and CLASS tables.



Here again, both the queries look alike, results alike. But when we compare the number of records and attributes involved at each stage, second query uses less records and hence more efficient.

1. Next step is to perform most restrictive joins and selection operations. When we say most restrictive joins and selection means, select those set of tables and views which will result in comparatively less number of records. Any query will have better performance when tables with few records are joined. Hence throughout heuristic method of optimization, the rules are formed to get less number of records at each stage, so that query performance is better. So is the case here too.

Suppose we have STUDENT, CLASS and TEACHER tables. Any student can attend only one class in an academic year and only one teacher takes a class. But a class can have more than 50 students. Now we have to retrieve STUDENT_NAME, ADDRESS, AGE, CLASS_NAME and TEACHER_NAME of each student in a school.

∏STD_NAME, ADDRESS, AGE, CLASS_NAME, TEACHER_NAME ((STUDENT ∞ CLASS_ID CLASS)∞ TECH_IDTEACHER)

Not So efficient

∏STD_NAME, ADDRESS, AGE, CLASS_NAME, TEACHER_NAME (STUDENT ∞ CLASS_ID (CLASS∞ TECH_IDTEACHER))

Efficient

In the first query, it tries to select the records of students from each class. This will result in a very huge intermediary table. This table is then joined with another small table. Hence the traversing of number of records is also more. But in the second query, CLASS and TEACHER are joined first, which has one to one relation here. Hence the number of resulting record is STUDENT table give the final result. Hence this second method is more efficient.

1. Sometimes we can combine above heuristic steps with cost based optimization technique to get better results.

All these methods need not be always true. It also depends on the table size, column size, type of selection, projection, join sort, constraints, indexes, statistics etc. Above optimization describes the best way of optimizing the queries.

2.1.5.7. Structure of Query Optimizer

We have seen only some of the many query processing strategies used in database systems. Most systems implement only a few strategies and, as a result, the number of strategies to be considered by the query optimizer is limited. Other systems consider a large number of strategies. For each strategy a cost estimate is computed.

In order to simplify the strategy selection task, a query may be split into several sub-queries. This not only simplifies strategy selection but also allows the query optimizer to recognize cases where a particular sub-query appear several times in the same query. By performing such subqueries only once, time is saved both in the query optimizing phase and in the execution of the query itself. Recognition of common sub-queries is analogous to the recognition of common sub-expressions in many optimizing compilers for programming languages. Clearly, examination of the query for common subs queries and the estimation of the cost of a large number of strategies impose a substantial overhead on query processing. However, the added cost of query optimization is usually more than offset by the savings at query execution time. Therefore, most commercial systems include relatively sophisticated optimizers.

2.1.5.8. Conclusion

There are a large number of possible strategies for processing a query, especially if the query is complex. Strategy selection can be done using information available in main memory, with little or no disk accesses. There are a large number of possible strategies for processing a query, especially if the query is complex. Strategy selection can be done using information available in main memory, with little or no disk accesses. The actual execution of the query will involve many accesses to disk. Since the transfer of data from disk is slow relative to the speed of main memory and the central processor of the computer system, it is advantageous to spend a considerable amount of processing to save disk accesses. Given a query, there are generally varieties of methods for computing the answer. It is the responsibility of the system to transform the query as entered by the user into an equivalent query, which can be computed more efficiently. The first action the system must take on a query is to translate the query into its internal form, which (for relational database systems) is usually based on the relational algebra. In the process of generating the internal form of the query, the parser checks the syntax of the user's query, verifies that the relations names appearing in the query are names of relation in the database, etc. If the query was expressed in terms of a view, the parser replaces all references to the view name with the relational algebra expression to compute the view.

2.1.5.9 Tutor Marked Assignment

1. At what point during query processing does optimization occur?
2. Why is it not desirable to force users to make an explicit choice of a query processing strategy? Are there cases in which it is desirable for users to be aware of the costs of competing? Query-processing strategies?
3. What is Query Optimization? Describe the stages of query optimization
4. Explain the various stages of query processing

2.1.7. References and Further Readings

Fegaras, L., Levine, D., Bose, S., & Chaluvadi, V. (2002, November). Query processing of streamed XML data. In *Proceedings of the eleventh international conference on Information and knowledge management* (pp. 126-133). ACM.

Kavanagh, T. S., Beall, C. W., Heiny, W. C., Motycka, J. D., Pendleton, S. S., Smallwood, T. D., ... & Traut, K. A. (1998). *U.S. Patent No. 5,838,965*. Washington, DC: U.S. Patent and Trademark Office.

Kim, W., Reiner, D. S., & Batory, D. (Eds.). (2012). *Query processing in database systems*. Springer Science & Business Media.

McHugh, J., Abiteboul, S., Goldman, R., Quass, D., & Widom, J. (1997). Lore: A database management system for semistructured data. *SIGMOD record*, *26*(3), 54-66.

Navathe, S. B., Tanaka, A. K., & Chakravarthy, S. (1992). Active Database Modeling and Design Tools: Issues, Approache, and Architecture. *IEEE Data Eng. Bull.*, *15*(1-4), 6-9.

Teorey, T. J. (1999). *Database modeling and design*. Morgan Kaufmann.

Teorey, T. J., Lightstone, S. S., Nadeau, T., & Jagadish, H. V. (2011). *Database modeling and design: logical design*. Elsevier.

MODULE 2: ADVANCED DATABASE FEATURES

UNIT 2: TRANSACTION MANAGEMENT AND RECOVERY

2.2.0. Introduction

Transaction processing systems are systems with large databases and hundreds of concurrent users. It provides and "all-or-noting" proposition stating that each work –unit performed in database must either complete in its entirety or have no effect whatsoever. Users of database systems are usually consider consistency and integrity of data as highly important. A simple transaction is usually issued to the database system in a language like SQL wrapped in a transaction. The concepts of Transaction management, types, properties and recovery strategies are described in this unit.

2.2.1. What is a Transaction?

A transaction is a sequence of read and write operations on data items that logically functions as one unit of work.

1. It should either be done entirely or not at all
2. If it succeeds, the effects of write operations persist (commit); if it fails, no effects of write operations persist (abort)
3. These guarantees are made despite concurrent activity in the system, and despite failures that may occur

2.2.1. 1 Properties of Transaction

There are four key properties of transactions that a DBMS must ensure to maintain data in the face of concurrent access and system failures:

1. Users should be able to regard the execution of each transaction as **atomic**: either all actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions (say, when a system crash occurs).
2. Each transaction, run by itself with no concurrent execution of other transactions, must preserve the consistency of the database. This property is called **consistency,**

and the DBMS assumes that it holds for each transaction. Ensuring this property of a transaction is the responsibility of the user.

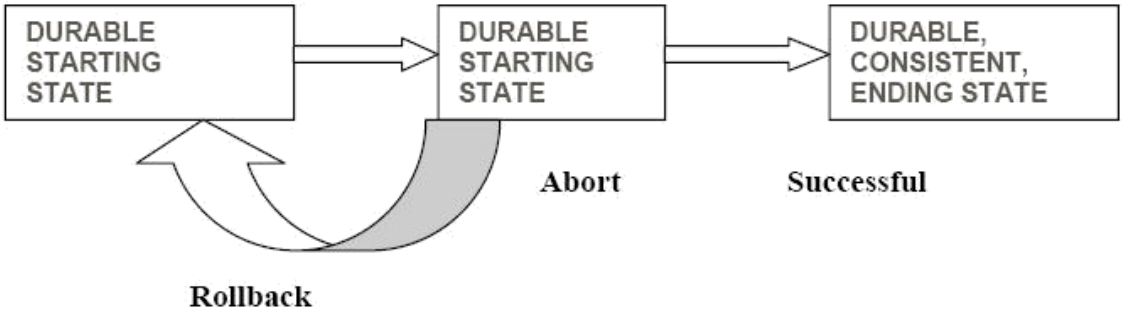


Figure 2.2.1: Database Transaction States

3. Users should be able to understand a transaction without considering the effect of other concurrently executing transactions, even if the DBMS interleaves the actions of several transactions for performance reasons. This property is sometimes referred to as **isolation**: Transactions are isolated, or protected, from the effects of concurrently scheduling other transactions.
4. Once the DBMS informs the user that a transaction has been successfully completed, its effects should persist even if the system crashes before all its changes are reflected on disk. This property is called **durability.**

The acronym **ACID** is sometimes used to refer to the four properties of transactions that we have presented here: atomicity, consistency, isolation and durability. We now consider how each of these properties is ensured in a DBMS.

1. **Consistency and Isolation:** Users are responsible for ensuring transaction consistency. That is, the user who submits a transaction must ensure that when run to completion by itself against a 'consistent' database instance, the transaction will leave the database in a 'consistent' state. For example, the user may have the consistency criterion that fund transfers between bank accounts should not change the total amount of money in the accounts. To transfer money from one account to another, a transaction must debit one account, temporarily leaving the database inconsistent in a global sense, even though the new account balance may satisfy any integrity constraints with respect to the range of acceptable account balances. The

user's notion of a consistent database is preserved when the second account is credited with the transferred amount. If a faulty transfer program always credits the second account with one naira less than the amount debited from the first account, the DBMS cannot be expected to detect inconsistencies due to such errors in the user program's logic.

1. The isolation property is ensured by guaranteeing that even though actions of several transactions might be interleaved, the net effect is identical to executing all transactions one after the other in some serial order. For example, if two transactions T1 and T2 are executed concurrently, the net effect is guaranteed to be equivalent to executing (all of) T1 followed by executing T2 or executing T2 followed by executing T1. (The DBMS provides no guarantees about which of these orders is effectively chosen.) If each transaction maps a consistent database instance to another consistent database instance, executing several transactions one after the other (on a consistent initial database instance) will also result in a consistent final database instance.
2. **Database consistency** is the property that every transaction sees a consistent database instance. Database consistency follows from transaction atomicity, isolation, and transaction consistency. Next, we discuss how atomicity and durability are guaranteed in a DBMS.

2. **Atomicity and Durability:** Transactions can be incomplete for three kinds of reasons. First, a transaction can be aborted, or terminated unsuccessfully, by the DBMS because some anomaly arises during execution. If a transaction is aborted by the DBMS for some internal reason, it is automatically restarted and executed anew. Second, the system may crash (e.g., because the power supply is interrupted) while one or more transactions are in progress. Third, a transaction may encounter an unexpected situation (for example, read an unexpected data value or be unable to access some disk) and decide to abort (i.e., terminate itself). Of course, since users think of transactions as being atomic, a transaction that is interrupted in the middle may leave the database in an inconsistent state. Thus a DBMS must find a way to remove the effects of partial transactions from the database, that is, it must ensure transaction atomicity: either all of a transaction's actions are carried out, or none are. A DBMS ensures transaction atomicity by undoing the actions of incomplete transactions. This means that users can ignore

incomplete transactions in thinking about how the database is modified by transactions over time. To be able to do this, the DBMS maintains a record, called the log, of all writes to the database. The log is also used to ensure durability: If the system crashes before the changes made by a completed transaction are written to disk, the log is used to remember and restore these changes when the system restarts. The DBMS component that ensures atomicity and durability is called the **recovery manager.**

2.2.1.2 Implementing Atomicity

Let's assume that before the transaction took place the balances in the account CA2090 is ₦50,000 and that in the account SB2359 is ₦35000. Now suppose that during the execution of the transaction, a failure (for example, a power failure) occurred that prevented the successful completion of the transaction. The failure occurred after the Write(CA2090); operation was executed, but before the execution of Write(SB2359); in this case the value of the accounts CA2090 and SB2359 are reflected in the database are ₦48,000 and ₦35000 respectively. The ₦2,000 that we have taken from the account is lost. Thus the failure has created a problem. The state of the database no longer reflects a real state of the world that the database is supposed to capture. Such a state is called an **inconsistent state.** The database system should ensure that such inconsistencies are not visible in a database system. It should be noted that even during the successful execution of a transaction there exists points at which the system is in an inconsistent state. But the difference in the case of a successful transaction is that the period for which the database is in an inconsistent state is very short and once the transaction is over the system will be brought back to a consistent state. So if a transaction never started or is completed successfully, the inconsistent states would not be visible except during the execution of the transaction. This is the reason for the atomicity requirement. If the atomicity property provided all actions of the transaction are reflected in the database of none are. The mechanism of maintaining atomicity is as follows The DBMS keeps tracks of the old values of any data on which a transaction performs a Write and if the transaction does not complete its execution, old values are restored o make it appear as though the

transaction never took place. The transaction management component of the DBMS ensures the atomicity of each transaction

2.2.1.23 Implementing Consistencies:

The consistency requirement in the above e. g is that the sum of CA2090 and SB2359 be unchanged by the execution of the transaction. Before the execution of the transaction the amounts in the accounts in CA2090 and SB2359 are ₦50,000 and ₦35,000 respectively. After the execution the amounts become ₦48,000 and ₦37,000. In both cases the sum of the amounts is 85,000 thus maintaining consistency. Ensuring the consistency for an individual transaction is the responsibility of the application programmer who codes the transaction.

2.2.1.24 Implementing the Isolation

Even if the atomicity and consistency properties are ensured for each transaction there can be problems if several transactions are executed concurrently. The different transactions interfere with one another and cause undesirable results. Suppose we are executing the above transaction T_i. We saw that the database is temporarily inconsistent while the transaction is being executed. Suppose that the transaction has performed the Write(CA2090) operation, during this time another transaction is reading the balances of different accounts. It checks the account CA2090 and finds the account balance at ₦48,000. Suppose that it reads the account balance of the other account (account SB2359, before the first transaction has got a chance to update the account. So the account balance in the account SB2359 is ₦35,000. After the second transaction has read the account balances, the first transaction reads the account balance of the account SB2359 and updates it to ₦37,000. But here we are left with a problem. The first transaction has executed successfully and the database is back to a consistent state. But while it was in an inconsistent state, another transaction performed some operations (May be updated the total account balances). This has left the database in an inconsistent state even after both the transactions have been executed successfully. On solution to the situation (concurrent execution of transactions) is to execute the transactions serially- one after the other. This can create many problems. Suppose long

transactions are being executed first. Then all other transactions will have to wait in the queue. There might be many transactions that are independent (or that do not interfere with one another). There is no need for such transactions to wait in the queue. Also concurrent executions of transactions have significant performance advantages. So the DBMS have found solutions to allow multiple transactions to execute concurrency without any problem. The isolation property of a transaction ensures that the concurrent execution of transactions result in a system state that is equivalent to a state that could have been obtained if the transactions were executed one after another. Ensuring isolation property is the responsibility of the concurrency-control component of the DBMS.

2.2.1.5 Implementing Durability

The durability property guarantees that, once a transaction completes successfully, all updates that it carried out on the database persist, even if there is a system failure after the transaction completes execution. We can guarantee durability by ensuring that either the updates carried out by the transaction have been written to the disk before the transaction completes or information about the updates that are carried out by the transaction and written to the disk are sufficient for the data base to reconstruct the updates when the data base is restarted after the failure. Ensuring durability is the responsibility of the recovery management component of the DBMS Picture Transaction management and concurrency control components of a DBMS.

2.2.2 Transaction States

Once a transaction is committed, we cannot undo the changes made by the transactions by rolling back the transaction. Only way to undo the effects of a committed transaction is to execute a compensating transaction. The creating of a compensating transaction can be quite complex and so the task is left to the user and it is not handled by the DBMS. The transaction must be in one of the following states:-

1. **Active:-** This is a initial state, the transaction stays in this state while it is executing

2. **Partially committed:** The transaction is in this state when it has executed the final statement
3. **Failed:** A transaction is in this state once the normal execution of the transaction cannot proceed.
4. **Aborted:** A transaction is said to be aborted when the transaction has rolled back and the database is being restored to the consistent state prior to the start of the transaction.
5. **Committed:** a transaction is in this committed state once it has been successfully executed and the database is transformed in to a new consistent state. Different transactions states are given in following figure 2.2.2.

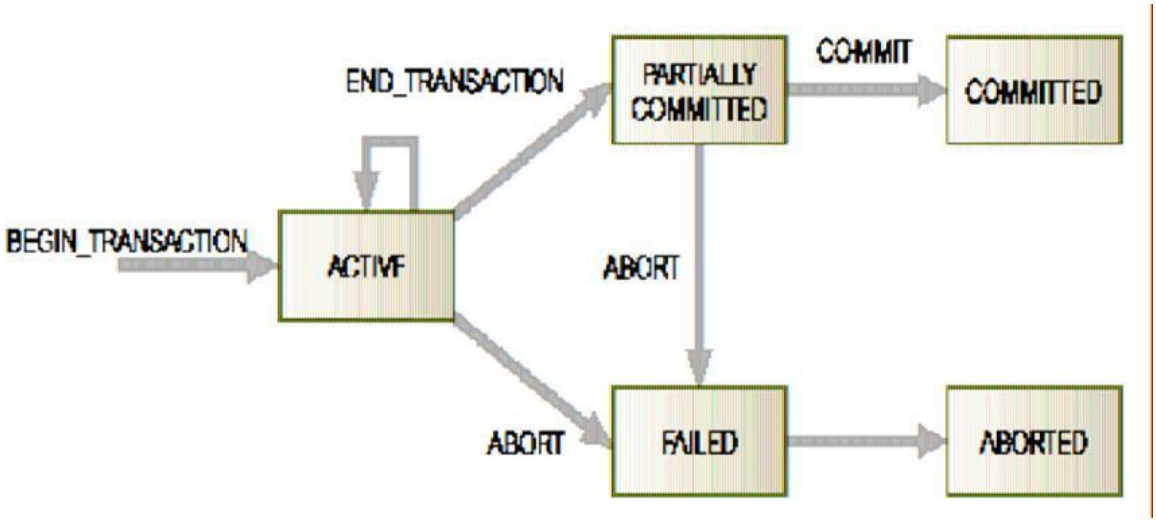


Figure 2.2.2: State Transition Diagram for a Transaction

2.2.3 Transactions and Schedules

A transaction is seen by the DBMS as a series, or list, of actions. The actions that can be executed by a transaction include reads and writes of database objects. A transaction can also be defined as a set of actions that are partially ordered. That is, the relative order of some of the actions may not be important. In order to concentrate on the main issues, we will treat transactions (and later, schedules) as a list of actions. Further, to keep our notation simple, we will assume that an object ***O*** is always read into a program variable that is also named ***O.*** We can therefore denote the action of a transaction ***T*** reading an object ***O*** as $R_T(O)$; similarly, we can denote writing as $W_T(O)$. When the transaction ***T*** is clear from the context, we will omit the subscript. In addition to reading and writing, each transaction must specify as its final action either commit (i.e., complete successfully) or abort (i.e., terminate and undo all the actions carried out thus far). **Abort** ***T*** denotes the action of ***T*** aborting, and ***Commit T*** denotes ***T*** committing.

A **schedule** is a list of actions (reading, writing, aborting, or committing) from a set of transactions, and the order in which two actions of a transaction ***T*** appear in a schedule must be the same as the order in which they appear in ***T***. Intuitively, a schedule represents an actual or potential execution sequence. For example, the schedule in Table 2.2.1.0shows an execution order for actions of two transactions ***T1*** and ***T2.*** We move forward in time as we go down from one row to the next. We emphasize that a schedule describes the actions of transactions as seen by the DBMS. In addition to these actions, a transaction may carry out other actions, such as reading or writing from operating system files, evaluating arithmetic expressions, and so on.

Table 2.2.1.0: Execution Order for Actions of two transactions

T1	**T2**
R(A)	
W(A)	
	R(B)
	W(B)
R(C)	
W(C)	

Notice that the schedule in Table 2.2.1.0 does not contain an abort or commit action for either transaction. A schedule that contains either an abort or a commit for each transaction whose actions are listed in it is called a **complete schedule**. A complete schedule must contain all the actions of every transaction that appears in it. If the actions of different transactions are not interleaved—that is, transactions are executed from start to finish, one by one—we call the schedule a **serial schedule.**

2.2.4 Concurrent Execution of Transactions

Now that we have introduced the concept of a schedule, we have a convenient way to describe interleaved executions of transactions. The DBMS interleaves the actions of different transactions to improve performance, in terms of increased throughput or improved response times for short transactions, but not all interleaving should be allowed.

2.2.4.1 Motivation for Concurrent Execution

The schedule shown in Table 2.2.1, represents an interleaved execution of the two transactions. Ensuring transaction isolation while permitting such concurrent execution is difficult, but is necessary for performance reasons. First, while one transaction is waiting for a page to be read in from disk, the CPU can process another transaction. This is because I/O activity can be done in parallel with CPU activity in a computer. Overlapping I/O and CPU activity reduces the amount of time disks and processors are idle, and increases **system throughput** (the average number of transactions completed in a given time). Second, interleaved execution of a short transaction with a long transaction usually allows the short transaction to complete quickly. In serial execution, a short transaction could get stuck behind a long transaction leading to unpredictable delays in **response time**, or average time taken to complete a transaction.

Serializability

To begin with, we assume that the database designer has defined some notion of a **consistent database state**. For example, we can define a consistency criterion for a university database to be that the sum of employee salaries in each department should be less than 80 percent of the budget for that department. We require that each transaction must **preserve** database consistency; it follows that any serial schedule that is complete will also preserve database consistency. That is, when a complete serial schedule is executed against a consistent database, the result is also a consistent database.

A **serializable schedule** over a set ***S*** of committed transactions is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete serial schedule over ***S.*** That is, the database instance that results from executing the given schedule is identical to the database instance that results from executing the transactions in some serial order.

1. There are some important points to note in this definition: Executing the transactions serially in different orders may produce different results, but all are presumed to be acceptable; the DBMS makes no guarantees about which of them will be the outcome of an interleaved execution.
2. The above definition of a serializable schedule does not cover the case of schedules containing aborted transactions.
3. If a transaction computes a value and prints it to the screen, this is an 'effect' that is not directly captured in the state of the database. We will assume that all such values are also written into the database, for simplicity.

2.2.4.1 Anomalies Associated with Interleaved Execution

We now illustrate three main ways in which a schedule involving two consistency preserving, committed transactions could run against a consistent database and leave it in an inconsistent state. Two actions on the same data object ***conflict*** if at least one of them is a **write**. The three anomalous situations can be described in terms of when the actions of two transactions ***T1*** and ***T2*** conflict with each other: in a **write - read *(WR)***

conflict ***T2*** reads a data object previously written by ***T1***; we define ***read-write (RW)*** and **write-write *(WW)*** conflicts similarly.

Reading Uncommitted Data (WR Conflicts)

The first source of anomalies is that a transaction ***T2*** could read a database object ***A*** that has been modified by another transaction ***T1,*** which has not yet committed. Such a read is called a dirty read. A simple example illustrates how such a schedule could lead to an inconsistent database state. Consider two transactions ***T1*** and ***T2***, each of which, run alone, preserves database consistency: **T1** transfers ₦100 from **A** to **B,** and ***T2*** increments both **A** and **B** by 6 percent (e.g., annual interest is deposited into these two accounts). Suppose that their actions are interleaved so that (1) the account transfer program **T1** deducts ₦100 from account **A**, then (2) the interest deposit program ***T2*** reads the current values of accounts **A** and **B** and adds 6 percent interest to each, and then (3) the account transfer program credits ₦100 to account **B**. The corresponding schedule, which is the view the DBMS has of this series of events, is illustrated in table 2. The result of this schedule is different from any result that

Table 2.2.2: Reading Uncommitted Data

T1	T2
R(A)	
W(A)	
R(B)	R(A)
W(B)	W(A)
Commit	R(B)
	W(B)
	Commit

we would get by running one of the two transactions first and then the other. The problem can be traced to the fact that the value of ***A*** written by ***T1*** is read by ***T2*** before ***T1*** has completed all its changes. The general problem illustrated here is that ***T1*** may write some value into A that makes the database inconsistent. As long as ***T1*** overwrites this value with a 'correct' value of A before committing, no harm is done if ***T1*** and ***T2*** run in some serial order, because ***T2*** would then not see the (temporary) inconsistency.

On the other hand, interleaved execution can expose this inconsistency and lead to an inconsistent final database state.

Note that although a transaction must leave a database in a consistent state after it completes, it is not required to keep the database consistent while it is still in progress. Such a requirement would be too restrictive: To transfer money from one account to another, a transaction must debit one account, temporarily leaving the database inconsistent, and then credit the second account, restoring consistency again.

1. Unrepeatable Reads (RW Conflicts)

The second way in which anomalous behavior could result is that a transaction ***T2*** could change the value of an object A that has been read by a transaction ***T1,*** while ***T1*** is still in progress. This situation causes two problems. First, if ***T1*** tries to read the value of **A** again, it will get a different result, even though it has not modified A in the meantime. This situation could not arise in a serial execution of two transactions; it is called an ***unrepeatable read***. Second, suppose that both ***T1*** and ***T2*** read the same value of A, say, 5, and then ***T1***, which wants to increment A by 1, changes it to 6, and ***T2,*** which wants to decrement A by 1, decrements the value that it read (that is, 5) and changes A to 4. Running these transactions in any serial order should leave A with a final value of 5; thus, the interleaved execution leads to an inconsistent state. The underlying problem here is that although ***T2's*** change is not directly read by ***T1***, it invalidates ***T1's*** assumption about the value of ***A,*** which is the basis for some of ***T1's*** subsequent actions.

2. Overwriting Uncommitted Data (WW Conflicts)

The third source of anomalous behavior is that a transaction ***T2*** could overwrite the value of an object A, which has already been modified by a transaction ***T1***, while ***T1*** is still in progress. Even if ***T2*** does not read the value of **A** written by ***T1***, a potential problem exists as the following example illustrates. Suppose that Harry and Larry are two employees, and their salaries must be kept equal. Transaction **T1** sets their salaries to ₦1, 000 and transaction ***T2*** sets their salaries

to ₦2,000. If we execute these in the serial order ***T1*** followed by ***T2***, both receive the salary ₦2,000; the serial order ***T2*** followed by ***T1*** gives each the salary ₦1,000. Either of these is acceptable from a consistency standpoint (although Harry and Larry may prefer a higher salary!). Notice that neither transaction reads a salary value before writing it—such a write is called a ***blind write***, for obvious reasons Now, consider the following interleaving of the actions of ***T1*** and ***T2***: ***T1*** sets Harry's salary to ₦1,000, T2 sets Larry's salary to ₦2,000, ***T1*** sets Larry's salary to ₦1,000, and finally ***T2*** sets Harry's salary to ₦2,000. The result is not identical to the result of either of the two possible serial executions, and the interleaved schedule is therefore not serializable. It violates the desired consistency criterion that the two salaries must be equal.

2.2.4.2 Schedules Involving Aborted Transactions

We now extend our definition of serializability to include aborted transactions. Intuitively, all actions of aborted transactions are to be undone, and we can therefore imagine that they were never carried out to begin with. Using this intuition, we extend the definition of a serializable schedule as follows: A serializable schedule over a set ***S*** of transactions is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete serial schedule over the set of *committed transactions in* ***S***.

This definition of serializability relies on the actions of aborted transactions being undone completely, which may be impossible in some situations. For example, suppose that (1) an account transfer program ***T1*** deducts ₦100 from account ***A,*** then (2) an interest deposit program ***T2*** reads the current values of accounts A and B and adds 6 percent interest to each, then commits, and then (3) ***T1*** is aborted. The corresponding schedule is shown in table 3. Now, ***T2*** has read a value for A that should never have been there (Recall that aborted transactions' effects are not supposed to be visible to other transactions.) If ***T2*** had not yet committed, we could deal with the situation by cascading the abort of ***T1*** and also aborting ***T2***; this process would recursively abort any transaction that read data written by ***T2,*** and so on. But ***T2*** has already committed, and

so we cannot undo its actions. We say that such a schedule is unrecoverable. A ***recoverable schedule*** is one in which transactions commit only after (and if!) all transactions whose changes they read commit. If transactions read only the changes of committed transactions, not only is the schedule recoverable, but also aborting a transaction can be accomplished without cascading the abort to other transactions. Such a schedule is said to **avoid cascading aborts.** There is another potential problem in undoing the actions of a transaction. Suppose that a transaction ***T2*** overwrites the value of an object A that has been modified by a transaction ***T1***, while ***T1*** is still in progress, and ***T1*** subsequently aborts. All of ***T1's*** changes to database objects are undone by restoring the value of any object that it modified to the value of the object before ***T1's*** changes. When **T1** is aborted, and its changes are undone in this manner, ***T2's*** changes are lost as well, even if ***T2*** decides to commit.

Table 2.2.3: An Unrecoverable Schedule

T1	T2
R(A) W(A)	
	R(A) W(A) R(B) W(B)
Abort	Commit

So, for example, if **A** originally had the value 5, then was changed by T1 to 6, and by T2 to 7, if ***T1*** now aborts, the value of **A** becomes 5 again. Even if ***T2*** commits, its change to A is inadvertently lost.

2.2.4.3 LOCK-BASED CONCURRENCY CONTROL

A DBMS must be able to ensure that only serializable, recoverable schedules are allowed, and that no actions of committed transactions are lost while undoing aborted transactions. A DBMS typically uses a locking protocol to achieve this. A locking protocol is a set of rules to be followed by each transaction (and enforced by the

DBMS), in order to ensure that even though actions of several transactions might be interleaved, the net effect is identical to executing all transactions in some serial order.

Strict Two-Phase Locking (Strict 2PL)

The most widely used locking protocol, called Strict Two-Phase Locking, or Strict 2PL, has two rules. The first rule is:

1. If a transaction T wants to read (respectively, modify) an object, it first requests a shared (respectively, exclusive) lock on the object.
2. All locks held by a transaction are released when the transaction is completed. Requests to acquire and release locks can be automatically inserted into transactions by the DBMS; users need not worry about these details.

In effect the locking protocol allows only 'safe' interleaving of transactions. If two transactions access completely independent parts of the database, they will be able to concurrently obtain the locks that they need and proceed merrily on their ways. On the other hand, if two transactions access the same object, and one of them wants to modify it, their actions are effectively ordered serially—all actions of one of these transactions (the one that gets the lock on the common object first) are completed before (this lock is released and) the other transaction can proceed. We denote the action of a transaction T requesting a shared (respectively, exclusive) lock on object O as $\boldsymbol{S_T\ (O)}$ (respectively, $\boldsymbol{X_T}$ (O)), and omit the subscript denoting the transaction when it is clear from the context. As an example, consider the schedule shown in table1. This interleaving could result in a state that cannot result from any serial execution of the three transactions. For instance, ***T1*** could change A from 10 to 20, then ***T2*** (which reads the value 20 for A) could change B from 100 to 200, and then T1 would read the value 200 for B. If run serially, either ***T1*** or **T2** would execute first, and read the values 10 for **A** and 100 for **B**: Clearly, the interleaved execution is not equivalent to either serial execution. If the Strict 2PL protocol is used, the above interleaving is disallowed. Let us see why. Assuming that the transactions proceed at the same relative speed as before, ***T1*** would obtain an exclusive lock on ***A*** first and then read and write A (table 4). Then, ***T2*** would request a lock on A. However, this request cannot be granted until

Table 2.2.5: Schedule Illustrating Strict 2PL

T1	T2
X(A) R(A) W(A)	

*T*1 releases its exclusive lock on *A*, and the DBMS therefore suspends *T*2. *T*1 now proceeds to obtain an exclusive lock on *B*, reads and writes *B*, then finally commits, at which time its locks are released. *T*2's lock request is now granted, and it proceeds. In this example the locking protocol results in a serial execution of the two transactions, shown in table 5. In general, however, the actions of different transactions could be interleaved. As an example, consider the interleaving of two transactions shown in table 4, which are permitted by the Strict 2PL protocol.

2.2.5 Concurrency control for database management

Concurrency control in database management systems permits many users (assumed to be interactive) to access a database in a multi programmed environment while preserving the illusion that each user has sole access to the system. Control is needed to coordinate concurrent accesses to a DBMS so that the overall correctness of the database is maintained. For example, users A and B both may wish to read and update the same record in the database at about the same time. The relative timing of the two transactions may have an impact on the state of the database at the end of the transactions. The end result may be an inconsistent database.

Functions of Concurrent Control

Several problems can occur when concurrent transactions execute in an uncontrolled manner.

– The lost update problem: This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of same database item incorrect.

– The temporary update (or dirty read) problem: This occurs when one transaction updates a database item and then the transaction fails for some reason. The

updated item is accessed by another transaction before it is changed back to its original value.

– The incorrect summary problem: If one transaction is calculating an aggregate function on a number of records while other transaction is updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

1. Whenever a transaction is submitted to a DBMS for execution, the system must make sure that :
   – All the operations in the transaction are completed successfully and their effect is recorded permanently in the database; or
   – the transaction has no effect whatever on the database or on the other transactions in the case of that a transaction fails after executing some of operations but before executing all of them.

Following are the problems created due to the concurrent execution of the transactions:

-Multiple update problems

In this problem, the data written by one transaction (an update operation) is being overwritten by another update transaction. This can be illustrated using our banking example. Consider our account CA2090 that has ₦50,000 balance in it. Suppose a transaction T1 is withdrawing ₦10,000 from the account while another transaction T2 is depositing ₦20,000 to the account. If these transactions were executed serially (one after another), the final balance would be ₦60,000, irrespective of the order in which the transactions are performed. In other words, if the transactions were performed serially, then the result would be the same if T1 is performed first or T2 is performed first-order is not important. But the transactions are performed concurrently, then depending on how the transactions are executed the results will vary. Consider the execution of the transactions given in Table 2.2.6

Table 2.2.6: Execution Transaction

Sequence	T1	T2	Account Balance
01	Begin Transaction		50000
02	Read (CA2090)	Begin Transaction	50000
03	CA2090:=CA2090-10000	Read (CA2090)	50000
04	Write (CA2090)	CA2090:=CA2090+20000	40000
05	Commit	Write (CA2090)	70000
06		Commit	70000

Both transactions start nearly at the same time and both read the account balance of ₦50000. Both transactions perform the operations that they are supposed to perform- T1 will reduce the amount by ₦10000and will write the result to the data base; T2 will increase the amount by ₦20000 and will write the amount to the database overwriting the previous update. Thus the account balance will gain additional ₦10000 producing a wrong result. If T2 were to start execution first, the result would have been ₦40000 and the result would have been wrong again. This situation could be avoided by preventing T2 from reading the value of the account balance until the update by T1 has been completed.

Incorrect Analysis Problem

Problems could arise even when a transaction is not updating the database. Transactions that read the database can also produce wrong result, if they are allowed to read the database when the database is in an inconsistent state. This problem is often referred to as dirty read or unrepeatable data. The problem of dirty read occurs when a transaction reads several values from the data base while another transactions are updating the values. Consider the case of the transaction that reads the account balances from all accounts to find the total amount in various account. Suppose that there are other transactions, which are updating the account balances-either reducing the amount (withdrawals) or increasing the amount (deposits). So when the first transaction reads the account balances and finds the totals, it will be wrong, as it might have read the account balances before the update in the case of some accounts

and after the updates in other accounts. This problem is solved by preventing the first transaction (the one that reads the balances) from reading the account balances until all the transactions that update the accounts are completed.

Inconsistent Retrievals

Consider two users A and B accessing a department database simultaneously. The user A is updating the database to give all employees a 5% salary raise while user B wants to know the total salary bill of a department. The two transactions interfere since the total salary bill would be changing as the first user updates the employee records. The total salary retrieved by the second user may be a sum of some salaries before the raise and others after the raise. Such a sum could not be considered an acceptable value of the total salary (the value before the raise or after the raise would be).

A	Time	B
Read Employee 100	1	-
-	2	Sum = 0.0
Update Salary	3	-
-	4	Read Employee 100
Write Employee 100	5	-
-	6	Sum = Sum + Salary
Read Employee 101	7	-
-	8	Read Employee 101
Update Salary	9	-
-	10	Sum = Sum + Salary
Write Employee 101	11	-

The problem illustrated in the last example is called the inconsistent retrieval anomaly. During the execution of a transaction therefore, changes made by another transaction that has not yet committed should not be visible since that data may not be consistent.

Uncommitted Dependency

Consider the following situation

A	Time	B
-	1	Read Q
-	2	-
-	3	Write A
Read Q	4	-
-	5	Read R
-	6	-
Write Q	7	-
-	8	Failure (rollback)
-	9	-

Transaction A reads the value of Q that was updated by transaction B but was never committed. The result of Transaction A writing Q therefore will lead to an inconsistent state of the database. Also if the transaction A doesn't write Q but only reads it, it would be using a value of Q which never really existed! Yet another situation would occur if the roll back happens after Q is written by transaction A. The roll back would restore the old value of Q and therefore lead to the loss of updated Q by transaction A. This is called the uncommitted dependency anomaly.

Serializability is a given set of interleaved transactions is said to be serial sable if and only if it produces the same results as the serial execution of the same transactions. It is an important concept associated with locking. It guarantees that the work of concurrently executing transactions will leave the database state as it would have been if these transactions had executed serially. This requirement is the ultimate criterion for database consistency and is the motivation for the two-phase locking protocol, which dictates that no new locks can be acquired on behalf of a transaction after the DBMS releases a lock held by that transaction. In practice, this protocol generally means that locks are held until commit time.

Serializability is the classical concurrency scheme. It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order. It assumes that all accesses to the database are done using read

and write operations. A schedule is called ``correct" if we can find a serial schedule that is ``equivalent" to it. Given a set of transactions $T_1...T_n$, two schedules S_1 and S_2 of these transactions are equivalent if the following conditions are satisfied:

1. ***Read-Write Synchronization***: If a transaction reads a value written by another transaction in one schedule, then it also does so in the other schedule.
2. ***Write-Write Synchronization*:** If a transaction overwrites the value of another transaction in one schedule, it also does so in the other schedule.
3. These two properties ensure that there can be no difference in the effects of the two schedules

2.2.5.1 Serializable schedule

A schedule is serial if, for every transaction T participating the schedule, all the operations of T are executed consecutively in the schedule. Otherwise it is called non-serial schedule.

1. Every serial schedule is considered correct; some non-serial schedules give erroneous results.
2. A schedule S of n transactions is serializable if it is equivalent to some serial schedule of the same n transactions; a non-serial schedule which is not equivalent to any serial schedule is not serializable.
3. The definition of two schedules considered “equivalent”:– result equivalent: producing same final state of the database (is not used)
    1. **conflict equivalent:** If the order of any two conflicting operations is the same in both schedules.
    2. **view equivalent:** If each read operation of a transaction reads the result of the same write operation in both schedules and the write operations of each transaction must produce the same results.
    3. **Conflict serializable:** if a schedule S is conflict equivalent to some serial schedule. we can reorder the non-conflicting operations S until we form the equivalent serial schedule, and S is a serializable schedule.

4. **View Serializability:** Two schedules are said to be view equivalent if the following three conditions hold. The same set of transactions participate in S and S'; and S and S' include the same operations of those transactions. A schedule S is said to be view serializable if it is view equivalent to a serial schedule.

2.2.5.3 LOCKING

In order to execute transactions in an interleaved manner it is necessary to have some form of concurrency control.

- This enables a more efficient use of computer resources.
- One method of avoiding problems is with the use of locks.
- When a transaction requires a database object it must obtain a lock.
  Locking is necessary in a concurrent environment to assure that one process does not retrieve or update a record that is being updated by another process. Failure to use some controls (locking), would result in inconsistent and corrupt data.

Locks enable a multi-user DBMS to maintain the integrity of transactions by isolating a transaction from others executing concurrently. Locks are particularly critical in write intensive and mixed workload (read/write) environments, because they can prevent the inadvertent loss of data or Consistency problems with reads. In addition to record locking, DBMS implements several other locking mechanisms to ensure the integrity of other data structures that provide shared I/O, communication among different processes in a cluster and automatic recovery in the event of a process or cluster failure. Aside from their integrity implications, locks can have a significant impact on performance. While it may benefit a given application to lock a large amount of data (perhaps one or more tables) and hold these locks for a long period of time, doing so inhibits concurrency and increases the likelihood that other applications will have to wait for locked resources.

LOCKING RULES

There are various locking rules that are applicable when a user reads or writes a data to a database. The various locking rules are -

- Any number of transactions can hold S-locks on an item
- If any transaction holds an X-lock on an item, no other transaction may hold any lock on the item
- A transaction holding an X-lock may issue a write or a read request on the data item
- A transaction holding an S-lock may only issue a read request on the data item

2.2.5.4 DEADLOCK

There lies a threat while writing or reading data onto a database with the help of available resources, it is nothing but the deadlock condition. In operating systems or databases, a situation in which two or more processes are prevented from continuing while each waits for resources to be freed by the continuation of the other. Any of a number of situations where two or more processes cannot proceed because they are both waiting for the other to release some resource.

A situation in which processes of a concurrent processor are waiting on an event which will never occur. A simple version of deadlock for a loosely synchronous environment arises when blocking reads and writes are not correctly matched. For example, if two nodes both execute blocking writes to each other at the same time, deadlock will occur since neither write can complete until a complementary read is executed in the other node.

2.2.6 Locking Techniques for Concurrency Control Based on Time Stamp Ordering

The timestamp method for concurrency control does not need any locks and therefore there are no deadlocks. Locking methods generally prevent conflicts by making transaction to walk. Timestamp methods do not make the transactions wait. Transactions involved in a conflict are simply rolled back and restarted. A timestamp is a unique identifier created by the DBMS that indicates the relative starting time of a transaction. Timestamps are generated either using the system clock (generating a timestamp when the transaction starts to execute) or by incrementing a logical counter every time a new transaction starts. Time stamping is the concurrency control protocol in which the fundamentals goal is to order transactions globally in such a way that older transactions get priority in the event of a conflict. In the Time stamping method, if a transaction attempts to read or write a data

item, then a read or write operation is allowed only if the last update on that data item was carried out by an older transaction. Otherwise the transaction requesting the read or write is restarted and given a new timestamp to prevent it from continually aborting and restarting. If the restarted transaction is not allowed a new timestamp and is allowed a new timestamp and is allowed to retain the old timestamp, it will never be allowed to perform the read or write, because by that some other transaction which has a newer timestamp than the restarted transaction might not be to commit due to younger transactions having already committed.

In addition to the timestamp for the transactions, data items are also assigned timestamps. Each data item contains a read-timestamp and write-timestamp. The read-timestamp contains the timestamp of the last transaction that read the item and the write-timestamp contains the timestamp of the last transaction that updated the item. For a transaction T the timestamp ordering protocol works as follows:

1. **Transactions T** requests to read the data item 'X' that has already been updated by a younger (later or one with a greater timestamp) transaction. This means that an earlier transactions is trying to read a data item that has been updated by a later transaction T is too late to read the previous outdated value and any other values it has acquired are likely to be inconsistent with the updated value of the data item. In this situation, the transaction T is aborted and restarted with a new timestamp.
2. In all other cases, the transaction is allowed to proceed with the read operation. The read-timestamp of the data item is updated with the timestamp of transaction T.
3. Transaction t requests to write (update) the data item 'X' that has already been read by a younger (later or one with the greater timestamp) transaction. This means that the younger transaction is already using the current value of the data item and it would be an error to update it now. This situation occurs when a transaction is late in performing the write and a younger transaction has already read the old value or written a new one. In this case the transaction T is aborted and is restarted with a new timestamp.

4. Transaction T asks to write the data item 'X' that has already been written by a younger transaction. This means that the transaction T is attempting to write an old or obsolete value of the data item. In this case also the transaction T is aborted and is restarted with a new timestamp.
5. In all other cases the transaction T is allowed to proceed and the write-timestamp of the data item is updated with the timestamp of transaction T.

The above scheme is called basic timestamp ordering. This scheme guarantees that the transactions are conflict serializable and the results are equivalent to a serial schedule in which the transactions are executed in chronological order by the timestamps. In other words, the results of a basic timestamps ordering scheme will be as same as when all the transactions were executed one after another without any interleaving.

One of the problems with basic timestamp ordering is that it does not guarantee recoverable schedules. A modification to the basic timestamp ordering protocol that relaxes the conflict Serializability can be used to provide greater concurrency by rejecting obsolete write operations. This extension is known as Thomas's write rule. Thomas's write rule modifies the checks for a write operation by transaction T as follows.

- When the transaction T requests to write the data item 'X' whose values has already been read by a younger transaction. This means that the order transaction (transaction T) is writing an obsolete value to the data item. In this case the write operation is ignored and the transaction (transaction T) is allowed to continue as if the write were performed. This principle is called the 'ignore obsolete write rule'. This rule allows for greater concurrency.
- In all other cases the transactions T is allowed to proceed and the write timestamp of transaction T.

2.2.6.1 **Multiversion Concurrency Control Techniques (Mvcc)**

The aim of Multi-Version Concurrency is to avoid the problem of Writers blocking Readers and vice-versa, by making use of multiple versions of data. The problem of

Writers blocking Readers can be avoided if Readers can obtain access to a previous version of the data that is locked by Writers for modification. The problem of Readers blocking Writers can be avoided by ensuring that Readers do not obtain locks on data. Multi-Version Concurrency allows Readers to operate without acquiring any locks, by taking advantage of the fact that if a Writer has updated a particular record, its prior version can be used by the Reader without waiting for the Writer to Commit or Abort. In a Multi-version Concurrency solution, Readers do not block Writers, and vice versa. While Multi-version concurrency improves database concurrency, its impact on data consistency is more complex.

2.2.6.3 Requirements of Multi-Version Concurrency Systems

As its name implies, multi-version concurrency relies upon multiple versions of data to achieve higher levels of concurrency. Typically, a DBMS offering multi-version concurrency (MVDB), needs to provide the following features:

1. The DBMS must be able to retrieve older versions of a row.
2. The DBMS must have a mechanism to determine which version of a row is valid in the context of a transaction. Usually, the DBMS will only consider a version that was committed prior to the start of the transaction that is running the query. In order to determine this, the DBMS must know which transaction created a particular version of a row, and whether this transaction committed prior to the starting of the current transaction.

2.2.6.4 Approaches to Multi-Version Concurrency

There are essentially two approaches to multi-version concurrency. The first approach is to store multiple versions of records in the database, and garbage collect records when they are no longer required. This is the approach adopted by PostgreSQL and Firebird/Interbase. The second approach is to keep only the latest version of data in the database, as in SVDB implementations, but reconstruct older versions of data dynamically as required by exploiting information within the Write Ahead Log. This is the approach taken by Oracle and MySQL/InnoDb.

2.2.6.4 Concept of Database Recovery Management

The **recovery manager** of a DBMS is responsible for ensuring transaction *atomicity* and *durability*. It ensures atomicity by undoing the actions of transactions that do not commit and durability by making sure that all actions of committed transactions survive **system crashes**, (e.g., a core dump caused by a bus error) and **media failures** (e.g., a disk is corrupted). When a DBMS is restarted after crashes, the recovery manager is given control and must bring the database to a consistent state. The recovery manager is also responsible for undoing the actions of an aborted transaction. To see what it takes to implement a recovery manager, it is necessary to understand what happens during normal execution.

Table 2.2.5. Schedule Illustrating Strict 2PL with Serial Execution

T1	T2
X(A)	
R(A)	
W(A)	
X(B)	
R(B)	
W(B)	X(A)
	R(A)
Commit	W(A)
	X(B)
	R(B)
	W(B)
	Commit

Table 2.2.6*:* Schedule Following Strict 2PL with Interleaved Actions

T1	T2
S(A)	
R(A)	
	S(*A*)
	R(*A*)
	X(*B*)
	R(*B*)
	W(*B*)
	Commit
X(*C*)	
R(*C*)	
W(*C*)	
Commit	

2.2.7 Database Recovery Management

The **transaction manager** of a DBMS controls the execution of transactions. Before reading and writing objects during normal execution, locks must be acquired (and released at some later time) according to a chosen locking protocol. For simplicity of exposition, we make the following assumption.

Atomic Writes: Writing a page to disk is an atomic action. This implies that the system does not crash while a write is in progress and is unrealistic. In practice, disk writes do not have this property, and steps must be taken during restart after a crash to verify that the most recent write to a given page was completed successfully and to deal with the consequences if not.

2.2.7.1 Stealing frames and forcing Pages

With respect to writing objects, two additional questions arise:

1. Can the changes made to an object *O* in the buffer pool by a transaction *T* be written to disk before *T* commits? Such writes are executed when another transaction wants to bring in a page and the buffer manager chooses to replace the page containing *O*; of course, this page must have been unpinned by *T*. If

such writes are allowed, we say that a **steal** approach is used. (Informally, the second transaction `steals' a frame from *T*.).

2. When a transaction commits, must we ensure that all the changes it has made to objects in the buffer pool are immediately forced to disk? If so, we say that a **force** approach is used.

From the standpoint of implementing a recovery manager, it is simplest to use a buffer manager with a no-steal, force approach. If no-steal is used, we don't have to undo the changes of an aborted transaction (because these changes have not been written to disk), and if force is used, we don't have to redo the changes of a committed transaction. if there is a subsequent crash (because all these changes are guaranteed to have been written to disk at commit time). However, these policies have important drawbacks. The no-steal approach assumes that all pages modified by ongoing transactions can be accommodated in the buffer pool, and in the presence of large transactions (typically run in batch mode, e.g., payroll processing), this assumption is unrealistic. The force approach results in excessive page I/O costs. If a highly used page is updated in succession by 20 transactions, it would be written to disk 20 times. With a no-force approach, on the other hand, the in memory copy of the page would be successively modified and written to disk just once, reflecting the effects of all 20 updates, when the page is eventually replaced in the buffer pool (in accordance with the buffer manager's page replacement policy).

For these reasons, most systems use a steal, no-force approach. Thus, if a frame is dirty and chosen for replacement, the page it contains is written to disk even if the modifying transaction is still active (*steal*); in addition, pages in the buffer pool that are modified by a transaction are not forced to disk when the transaction commits (*no-force*).

2.2.7.1 Recovery-Related steps during normal execution

The recovery manager of a DBMS maintains some information during normal execution of transactions in order to enable it to perform its task in the event of a failure. In particular, a **log** of all modifications to the database is saved on **stable storage**, which is guaranteed (with very high probability) to survive crashes and media failures. Stable

storage is implemented by maintaining multiple copies of information (perhaps in different locations) on nonvolatile storage devices such as disks or tapes. It is important to ensure that the log entries describing a change to the database are written to stable storage *before* the change is made; otherwise, the system might crash just after the change, leaving us without a record of the change.

The log enables the recovery manager to undo the actions of aborted and incomplete transactions and to redo the actions of committed transactions. For example, a transaction that committed before the crash may have made updates to a copy (of a database object) in the buffer pool, and this change may not have been written to disk before the crash, because of a no-force approach. Such changes must be identified using the log, and must be written to disk. Further, changes of transactions that did not commit prior to the crash might have been written to disk because of a steal approach. Such changes must be identified using the log and then undone.

The **recovery manager** of a DBMS is responsible for ensuring two important properties of transactions: *atomicity* and *durability*. It ensures atomicity by undoing the actions of transactions that do not commit and durability by making sure that all actions of committed transactions survive **system crashes**, (e.g., a core dump caused by a bus error) and **media failures** (e.g., a disk is corrupted). The recovery manager is one of the hardest components of a DBMS to design and implement. It must deal with a wide variety of database states because it is called on during system failures. In this unit, we present the **ARIES** recovery algorithm, which is conceptually simple, works well with a wide range of concurrency control mechanisms, and is being used, in an increasing number of database systems. We begin with an introduction to ARIES.

2.2.8 Introduction to ARIES

ARIES is a recovery algorithm that is designed to work with a steal, no-force approach. When the recovery manager is invoked after a crash, restart proceeds in three phases:

1. **Analysis:** Identifies dirty pages in the buffer pool (i.e., changes that have not been written to disk) and active transactions at the time of the crash.

2. **Redo:** Repeats all actions, starting from an appropriate point in the log, and restores the database state to what it was at the time of the crash.
3. **Undo:** Undoes the actions of transactions that did not commit, so that the database reflects only the actions of committed transactions.

There are three main principles behind the ARIES recovery algorithm:

1. **Write-ahead logging:** Any change to a database object is first recorded in the log; the record in the log must be written to stable storage before the change to the database object is written to disk.
2. **Repeating history during Redo:** Upon restart following a crash, ARIES retraces all actions of the DBMS before the crash and brings the system back to the exact state that it was in at the time of the crash. Then, it undoes the actions of transactions that were still active at the time of the crash (effectively aborting them).
3. **Logging changes during Undo:** Changes made to the database while undoing a transaction are logged in order to ensure that such an action is not repeated in the event of repeated (failures causing) restarts.

The second point distinguishes ARIES from other recovery algorithms and is the basis for much of its simplicity and flexibility. In particular, ARIES can support concurrency control protocols that involve locks of finer granularity than a page (e.g., record-level locks). The second and third points are also important in dealing with operations such that redoing and undoing the operation are not exact inverses of each other.

The Log

The log, sometimes called the **trail** or **journal**, is a history of actions executed by the DBMS. Physically, the log is a _le of records stored in stable storage, which is assumed to survive crashes; this durability can be achieved by maintaining two or more copies of the log on different disks (perhaps in different locations), so that the chance of all copies of the log being simultaneously lost is negligibly small. The most recent portion of the log, called the **log tail**, is kept in main memory and is periodically forced to stable

storage. This way, log records and data records are written to disk at the same granularity (pages or sets of pages).

1. Every **log record** is given a unique *id* called the **log sequence number (LSN)**. As with any record id, we can fetch a log record with one disk access given the LSN. Further, LSNs should be assigned in monotonically increasing order; this property is required for the ARIES recovery algorithm. If the log is a sequential file, in principle growing indefinitely, the LSN can simply be the address of the first byte of the log record. For recovery purposes, every page in the database contains the LSN of the most recent log record that describes a change to this page. This LSN is called the **pageLSN**. A log record is written for each of the following actions:

1. **Updating a page**: After modifying the page, an *update* type record is appended to the log tail. The pageLSN of the page is then set to the LSN of the update log record. (The page must be pinned in the buffer pool while these actions are carried out.)

2. **Commit**: When a transaction decides to commit, it **force-writes** a *commit* type log record containing the transaction id. That is, the log record is appended to the log, and the log tail is written to stable storage, up to and including the commit record. The transaction is considered to have committed at the instant that its commit log record is written to stable storage.

3. **Abort**: When a transaction is aborted, an *abort* type log record containing the transaction id is appended to the log, and Undo is initiated for this transaction

1. **End**: As noted above, when a transaction is aborted or committed, some additional actions must be taken beyond writing the abort or commit log record. After all these additional steps are completed, an *end* type log record containing the transaction id is appended to the log.

2. **Undoing an update:** When a transaction is rolled back (because the transaction is aborted, or during recovery from a crash), its updates are undone. When the action described by an update log record is undone, a *compensation log record*, or CLR, is written.

3. Every log record has certain fields: **prevLSN**, **transID**, and **type**. The set of all log records for a given transaction is maintained as a linked list going back in time, using the **prevLSN** field; this list must be updated whenever a log record is added. The transID field is the id of the transaction generating the log record, and the type field obviously indicates the type of the log record. Additional fields depend on the type of the log record. We have already mentioned the additional contents of the various log record types, with the exception of the update and compensation log record types, which we describe next.

2.2.8.1 Update Log Records

The fields in an **update** log record are illustrated in table 2.2.7. The **pageID** field

Table 2.2.7: Contents of an Update Log Record

prevLSN	transID	type	pageID	length	offset	before-offset	after-offset

Fields common to all log

Additional fields for update log records

is the page id of the modified page; the length in bytes and the offset of the change are also included. The **before-image** is the value of the changed bytes before the change; the **after-image** is the value after the change. An update log record that contains both before- and after-images can be used to redo the change and to undo it. In certain contexts, which we will not discuss further, we can recognize that the change will never be undone (or, perhaps, redone). A **redo-only update** log record will contain just the after-image; similarly an **undo-only update** record will contain just the before-image.

Compensation Log Records

A **compensation log record (CLR)** is written just before the change recorded in an update log record *U* is undone. (Such an undo can happen during normal system execution when a transaction is aborted or during recovery from a crash.) A compensation log record *C* describes the action taken to undo the actions recorded in the

corresponding update log record and is appended to the log tail just like any other log record. The compensation log record *C* also contains a field called **undoNextLSN**, which is the LSN of the next log record that is to be undone for the transaction that wrote update record *U*; this field in *C* is set to the value of prevLSN in *U*.As an example, consider the fourth update log record shown in Figure 20.3. If this update is undone, a CLR would be written, and the information in it would include the transID, pageID, length, offset, and before-image fields from the update record. Notice that the CLR records the (undo) action of changing the affected bytes back to the before-image value; thus, this value and the location of the affected bytes constitute the redo information for the action described by the CLR.

Unlike an update log record, a CLR describes an action that will never be *undone*, that is, we never undo an undo action. The reason is simple: an update log record describes a change made by a transaction during normal execution and the transaction may subsequently be aborted, whereas a CLR describes an action taken to rollback a transaction for which the decision to abort has already been made. Thus, the transaction *must* be rolled back, and the undo action described by the CLR is definitely required. This observation is very useful because it bounds the amount of space needed for the log during restart from a crash: The number of CLRs that can be written during Undo is no more than the number of update log records for active transactions at the time of the crash. It may well happen that a CLR is written to stable storage (following WAL, of course) but that the undo action that it describes is not yet written to disk when the system crashes again. In this case the undo action described in the CLR is reapplied during the Redo phase, just like the action described in update log records. For these reasons, a CLR contains the information needed to reapply, or redo, the change described but not to reverse it.

2.2.8.3 Other Recovery-Related Data Structures

In addition to the log, the following two tables contain important recovery-related information:

1. **Transaction table:** This table contains one entry for each active transaction. The entry contains (among other things) the transaction id, the status, and a field called **lastLSN**, which is the LSN of the most recent log record for this transaction. The **status** of a transaction can be that it is in progress, is committed, or is aborted. (In the latter two cases, the transaction will be removed from the table once certain `clean up' steps are completed.)

   Dirty page table: This table contains one entry for each dirty page in the buffer pool, that is, each page with changes that are not yet reflected on disk. The entry contains a field **recLSN**, which is the LSN of the first log record that caused the page to become dirty. Note that this LSN identifies the earliest log record that might have to be redone for this page during restart from a crash. During normal operation, these are maintained by the transaction manager and the buffer manager, respectively, and during restart after a crash, these tables are reconstructed in the Analysis phase of restart.

   The Write-Ahead Log Protocol

   Before writing a page to disk, every update log record that describes a change to this page must be forced to stable storage. This is accomplished by forcing all log records up to and including the one with LSN equal to the pageLSN to stable storage before writing the page to disk.

The importance of the WAL protocol cannot be over emphasized WAL is the fundamental rule that ensures that a record of every change to the database is available while attempting to recover from a crash. If a transaction made a change and committed, the no-force approach means that some of these changes may not have been written to disk at the time of a subsequent crash. Without a record of these changes, there would be no way to ensure that the changes of a committed transaction survive crashes. Note that the definition of a *committed transaction* is effectively \a transaction whose log records, including a commit record, have all been written to stable storage"! When a transaction is committed, the log tail is forced to stable storage, even if a no force approach is being used. It is worth contrasting this operation with the actions taken

under a force approach: If a force approach is used, all the pages modified by the transaction, rather than a portion of the log that includes all its records, must be forced to disk when the transaction commits.

The set of all changed pages is typically much larger than the log tail because the size of an update log record is close to (twice) the size of the changed bytes, which is likely to be much smaller than the page size. Further, the log is maintained as a sequential file, and thus all writes to the log are sequential writes. Consequently, the cost of forcing the log tail is much smaller than the cost of writing all changed pages to disk.

Checkpointing

A **checkpoint** is like a snapshot of the DBMS state, and by taking checkpoints periodically, as we will see, the DBMS can reduce the amount of work to be done during restart in the event of a subsequent crash. *Checkpointing* in ARIES has three steps. First, a **begin checkpoint** record is written to indicate when the checkpoint starts. Second, an **end checkpoint** record is constructed, including in it the current contents of the transaction table and the dirty page table, and appended to the log. The third step is carried out after the **end checkpoint** record is written to stable storage: A special **master** record containing the LSN of the *begin checkpoint* log record is written to a known place on stable storage. While the end checkpoint record is being constructed, the DBMS continues executing transactions and writing other log records; the only guarantee we have is that the transaction table and dirty page table are accurate *as of the time of the begin checkpoint record*.

This kind of checkpoint is called a **fuzzy checkpoint** and is inexpensive because it does not require quiescing the system or writing out pages in the buffer pool (unlike some other forms of checkpointing). On the other hand, the effectiveness of this checkpointing technique is limited by the earliest recLSN of pages in the dirty pages table, because during restart we must redo changes starting from the log record whose LSN is equal to this recLSN. Having a background process that periodically writes dirty

pages to disk helps to limit this problem. When the system comes back up after a crash, the restart process begins by locating the most recent checkpoint record. For uniformity, the system always begins normal execution by taking a checkpoint, in which the transaction table and dirty page table are both empty.

2.2.8.4 Recovering from a System Crash

When the system is restarted after a crash, the recovery manager proceeds in three Phases. The Analysis phase begins by examining the most recent begin checkpoint record, whose LSN is denoted as C and proceeds forward in the log until the last log record. The Redo phase follows Analysis and redoes all changes to any page that might have been dirty at the time of the crash; this set of pages and the starting point for Redo (the smallest recLSN of any dirty page) are determined during Analysis. The Undo phase follows Redo and undoes the changes of all transactions that were active at the time of the crash; again, this set of transactions is identified during the Analysis phase. Notice that Redo reapplies changes in the order in which they were originally carried out; Undo reverses changes in the opposite order, reversing the most recent change first.

Analysis Phase

The **Analysis** phase performs three tasks:

2. It determines the point in the log at which to start the Redo pass.
3. It determines (a conservative superset of the) pages in the buffer pool that were dirty at the time of the crash.
4. It identifies transactions that were active at the time of the crash and must be undone.

Analysis begins by examining the most recent begin checkpoint log record and initializing the dirty page table and transaction table to the copies of those structures in the next end checkpoint record. Thus, these tables are initialized to the set of dirty pages and active transactions at the time of the checkpoint. (If there are additional log records between the begin checkpoint and end checkpoint records, the tables must be adjusted to

reflect the information in these records, but we omit the details of this Analysis then scans the log in the forward direction until it reaches the end of the log:

1. If an end log record for a transaction *T* is encountered, *T* is removed from the transaction table because it is no longer active.
2. If a log record other than an end record for a transaction *T* is encountered, an entry for *T* is added to the transaction table if it is not already there. Further, the entry for *T* is modified:
    1. The lastLSN field is set to the LSN of this log record.
    2. If the log record is a commit record, the status is set to C, otherwise it is set to U (indicating that it is to be undone).

1. If a redo able log record affecting page *P* is encountered, and *P* is not in the dirty page table, an entry is inserted into this table with page id *P* and recLSN equal to the LSN of this re doable log record. This LSN identifies the oldest change affecting page *P* that may not have been written to disk.

At the end of the Analysis phase, the transaction table contains an accurate list of all transactions that were active at the time of the crash | this is the set of transactions with status U. The dirty page table includes all pages that were dirty at the time of the crash, but may also contain some pages that were written to disk. If an *end write* log record were written at the completion of each write operation, the dirty page table constructed during Analysis could be made more accurate, but in ARIES, the additional cost of writing end write log records is not considered to be worth the gain.

2.2.8.4 Redo Phase

1. During the **Redo** phase, ARIES reapplies the updates of *all* transactions, committed or otherwise. Further, if a transaction was aborted before the crash and its updates were undone, as indicated by CLRs, the actions described in the CLRs are also reapplied.

2. This **repeating history** paradigm distinguishes ARIES from other proposed WALbased recovery algorithms and causes the database to be brought to the same state that it was in at the time of the crash.
3. The Redo phase begins with the log record that has the smallest recLSN of all pages in the dirty page table constructed by the Analysis pass because this log record identifies the oldest update that may not have been written to disk prior to the crash. Starting from this log record, Redo scans forward until the end of the log. For each redo able log record (update or CLR) encountered, Redo checks whether the logged action must be redone. The action must be redone unless one of the following conditions holds:

1. The affected page is not in the dirty page table, or
2. The affected page is in the dirty page table, but the recLSN for the entry is *greater than* the LSN of the log record being checked, or
3. The pageLSN (stored on the page, which must be retrieved to check this condition) is *greater than or equal* to the LSN of the log record being checked.

1. The first condition obviously means that all changes to this page have been written to disk. Because the recLSN is the first update to this page that may not have been written to disk, the second condition means that the update being checked was indeed propagated to disk. The third condition, which is checked last because it requires us to retrieve the page, also ensures that the update being checked was written to disk because either this update or a later update to the page was written. (Recall our assumption that a write to a page is atomic; this assumption is important here!) If the logged action must be redone:
2. The logged action is reapplied.
3. The pageLSN on the page is set to the LSN of the redone log record. No additional log record is written at this time.

2.2.8.5 Undo Phase

The Undo phase, unlike the other two phases, scans backward from the end of the log. The goal of this phase is to undo the actions of all transactions that were active at the time of the crash, that is, to effectively abort them. This set of transactions is identified in the transaction table constructed by the Analysis phase.

The Undo Algorithm

Undo begins with the transaction table constructed by the Analysis phase, which identifies all transactions that were active at the time of the crash, and includes the LSN of the most recent log record (the lastLSN field) for each such transaction. Such transactions are called **loser transactions**. All actions of losers must be undone, and further, these actions must be undone in the reverse of the order in which they appear in the log.

Consider the set of lastLSN values for all loser transactions. Let us call this set **ToUndo**. Undo repeatedly chooses the largest (i.e., most recent) LSN value in this set and processes it, until ToUndo is empty. To process a log record:

1. If it is a CLR, and the undoNextLSN value is not *null*, the undoNextLSN value is added to the set ToUndo; if the undoNextLSN is *null*, an end record is written for the transaction because it is completely undone, and the CLR is discarded.
2. If it is an update record, a CLR is written and the corresponding action is undone and the prevLSN value in the update log record is added to the set ToUndo. When the set ToUndo is empty, the Undo phase is complete. Restart is now complete,and the system can proceed with normal operations.

Aborting a Transaction

Aborting a transaction is just a special case of the Undo phase of Restart in which a single transaction, rather than a set of transactions is undone.

2.2.9 Media Recovery

Media recovery is based on periodically making a copy of the database. Because copying a large database object such as a file can take a long time, and the DBMS must be allowed to continue with its operations in the meantime, creating a copy is handled in a manner similar to taking a fuzzy checkpoint.

When a database object such as a file or a page is corrupted, the copy of that object is brought up-to-date by using the log to identify and reapply the changes of committed transactions and undo the changes of uncommitted transactions (as of the time of the media recovery operation). The begin checkpoint LSN of the most recent complete checkpoint is recorded along with the copy of the database object in order to minimize the work in reapplying changes of committed transactions. Let us compare the smallest recLSN of a dirty page in the corresponding end checkpoint record with the LSN of the begin checkpoint record and call the smaller of these two LSNs *I*.

We observe that the actions recorded in all log records with LSNs less than *I* must be reflected in the copy. Thus, only log records with LSNs greater than *I* need to be reapplied to the copy. Finally, the updates of transactions that are incomplete at the time of media recovery or that were aborted after the fuzzy copy was completed need to be undone to ensure that the page reflects only the actions of committed transactions. The set of such transactions can be identified as in the Analysis pass.

2.2.9.1 Shadow Paging

This recovery scheme does not require the use of a log in a single-user environment. In a multiuser environment, a log may be needed for the concurrency control method. Shadow paging considers the database to be made up of a number of fixed-size disk pages (or disk blocks)-say, n-for recovery purposes. A directory with n entries' is constructed, where the i^{th} entry points to the i^{th} database page on disk. The directory is kept in main memory if it is not too large, and all references-reads or writes-to database pages on disk go through it. When a transaction begins executing, the current directory-whose entries point to the most recent or current database pages on disk-is copied into a shadow directory. The shadow directory is then saved on disk while the current directory is used by the transaction. During transaction execution, the shadow directory

is *never* modified. When a write_ item operation is performed, a new copy of the modified database page is created, but the old copy of that page is *not overwritten.* Instead, the new page is written elsewhere-on some previously unused disk block. The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block

Committing a transaction corresponds to discarding the previous shadow directory. Since recovery involves neither undoing nor redoing data items, this technique can be categorized as a NO-UNDO/NO-REDO technique for recovery. In a multiuser environment with concurrent transactions, logs and checkpoints must be incorporated into the shadow paging technique. One disadvantage of shadow paging is that the updated database pages change location on disk. This makes it difficult to keep related database pages close together on disk without complex storage management strategies. Furthermore, if the directory is large, the overhead of writing shadow directories to disk as transactions commit is significant. A further complication is how to handle garbage collection when a transaction commits. The old pages referenced by the shadow directory that have been updated must be released and added to a list of free pages for future use. These pages are no longer needed after the transaction commits. Another issue is that the operation to migrate between current and shadow directories must be implemented as an atomic operation.

2.2.9.2 Other algorithms and interaction with Concurrency Control

Like ARIES, the most popular alternative recovery algorithms also maintain a log of database actions according to the WAL protocol. A major distinction between ARIES and these variants is that the Redo phase in ARIES *repeats history*, that is, redoes the actions of *all* transactions, not just the non-losers. Other algorithms redo only the non-losers, and the Redo phase follows the Undo phase, in which the actions of losers are rolled back.

Repeating history paradigm and the use of CLRs, ARIES is able to support fine-granularity locks (record-level locks) and logging of logical operations, rather than just byte-level modifications. For example, consider a transaction *T* that inserts a data entry 15_* into a B+ tree index. Between the times this insert is done and the time that *T* is

eventually aborted, other transactions may also insert and delete entries from the tree. If record-level locks are set, rather than page-level locks, it is possible that the entry $15*$ is on a different physical page when *T* aborts from the one that *T* inserted it into. In this case the undo operation for the insert of $15*$ must be recorded in logical terms because the physical (byte-level) actions involved in undoing this operation are not the inverse of the physical actions involved in inserting the entry.

Logging logical operations yields considerably higher concurrency, although the use of fine-granularity locks can lead to increased locking activity (because more locks must be set). Thus, there is a trade-off between different WAL-based recovery schemes. We have chosen to cover ARIES because it has several attractive properties, in particular, its simplicity and its ability to support fine-granularity locks and logging of logical operations.

One of the earliest recovery algorithms, used in the System R prototype at IBM, takes a very different approach. There is no logging and, of course, no WAL protocol. Instead, the database is treated as a collection of pages and accessed through a **page table**, which maps page ids to disk addresses. When a transaction makes changes to a data page, it actually makes a copy of the page, called the **shadow** of the page, and changes the shadow page. The transaction copies the appropriate part of the page table and changes the entry for the changed page to point to the shadow, so that it can see the changes; however, other transactions continue to see the original page table, and therefore the original page, until this transaction commits. Aborting a transaction is simple: just discard its shadow versions of the page table and the data pages. Committing a transaction involves making its version of the page table public and discarding the original data pages that are superseded by shadow pages.

This scheme suffers from a number of problems. First, data becomes highly fragmented due to the replacement of pages by shadow versions, which may be located far from the original page. This phenomenon reduces data clustering and makes good garbage collection imperative. Second, the scheme does not yield a sufficiently high degree of concurrency. Third, there is a substantial storage overhead due to the use of

shadow pages. Fourth, the process aborting a transaction can itself run into deadlocks, and this situation must be specially handled because the semantics of aborting an abort transaction gets murky. For these reasons, even in System R, shadow paging was eventually superseded by WAL-based recovery techniques.

2.2.9.3 Database Backup and Recovery from Catastrophic Failures

So far, all the techniques we have discussed apply to non-catastrophic failures. A key assumption has been that the system log is maintained on the disk and is not lost as a result of the failure. Similarly, the shadow directory must be stored on disk to allow recovery when shadow paging is used. The recovery techniques we have discussed use the entries in the system log or the shadow directory to recover from failure by bringing the database back to a consistent state.

The recovery manager of a DBMS must also be equipped to handle more catastrophic failures such as disk crashes. The main technique used to handle such crashes is that of database backup. The whole database and the log are periodically copied onto a cheap storage medium such as magnetic tapes. In case of a catastrophic system failure, the latest backup copy can be reloaded from the tape to the disk, and the system can be restarted. To avoid losing all the effects of transactions that have been executed since the last backup, it is customary to back up the system log at more frequent intervals than full database backup by periodically copying it to magnetic tape. The system log is usually substantially smaller than the database itself and hence can be backed up more frequently. Thus users do not lose all transactions they have performed since the last database backup. All committed transactions recorded in the portion of the system log that has been backed up to tape can have their effect on the database redone. A new log is started after each database backup. Hence, to recover from disk failure, the database is first recreated on disk from its latest backup copy on tape.

1. Self-Assessment Exercises

1. How does the recovery manager ensure atomicity of transactions?
2. How does it ensure durability?
3. What is the difference between stable storage and disk?
4. What is the difference between a system crash and a media failure?
5. Explain the WAL protocol.
6. Describe the steal and no-force policies

2.2.11 Tutor Marked Assignment

1. What are the properties required of LSNs?
2. What are the fields in an update log record? Explain the use of each field.
3. What are redo able log records?
4. What are the differences between update log records and CLRs?

2.2.12 References/Suggested Readings

1 Date, C.J., Introduction to Database Systems (7th Edition) Addison Wesley, 2000
2 Leon, Alexis and Leon, Mathews, Database Management Systems, LeonTECHWorld
3 Elamasri R . and Navathe, S., Fundamentals of Database Systems (3 Edition), Pearsson Education, 2000.

Module 2: Advanced Database Features

Unit 3: Database Security & Authorization

2.3.0 INTRODUCTION

A common problem of security for all computer systems is to prevent unauthorized persons from gaining access to the system, either for information, making malicious changes to all or a portion or entire database. In this unit, we examine the ways in which data may become inconsistent or be misused. We then present mechanisms to guard against their occurrence.

2.3.1 OBJECTIVES

At the end of this unit, students should be able to:

1. understand the various security and authorization concepts and forms
2. understand the various constraints in ensuring database consistency
1. understand and apply appropriate security measures

2.3.2 DATABASE SECURITY AND AUTHORIZATION: Basic Concepts

The Databases not protected are the dream of cyber-criminal. Those databases contain valuable data of the organization could be easy target of an attack and are conveniently organized. It is not surprising that the databases are the main target of sophisticated cyber-attacks crackers2 and, increasingly, users working in the organization and have privileges. However, there are many steps you take to protect databases in your organization and at the same time, its reputation.

The major goal of an attacker is to obtain vital information about an individual, government, organization or any establishment of their interest without the authorization of the owner. Information gotten will be used for fraudulent activities. Security mechanism such as access control, encryption should be enforced to protect the system from unauthorized access. Security mechanisms can be oriented access control policies based on user identity, known as discretionary safety or policies that restrict

access to information classified as confidential to authorized personnel, called Mandatory Safety.

2.3.3.1 *What is Database Security?*

Database security is the collective measures used to protect and secure database or database management software from illegitimate use, malicious threats and attacks.

2.3.3.2 *What is Database Authorization?*

It is the granting and denying of permission to an authorized user to carry out particular transactions, and hence to change the state of the database (write item transactions) and/or receive data from the database (read-item transactions).

2.3.3.3 *What Is Database Security And Authorization?*

Database security and authorization is the act of protecting the database from unauthorized access using security measures as well as granting and denying privileges to authorized users. The relationship is that both Database Security and Database Authorization are used for protecting the database from unauthorized access. The difference is that database Security is mostly used to protect database externally while authorization is used to protect the database internally.

2.3.4 *Types of Security Breaches in A Database System*

Database security is a very broad area that addresses many issues, including the following:

1. **Legal and ethical issues** regarding the right to access certain information. Some information may be deemed to be private and cannot be accessed legally by unauthorized persons.
2. Policy issues at the governmental, institutional, or corporate level as to what kinds of information should not be made publicly available-for example, credit ratings and personal medical records.
1. System-related issues such as the *system levels* at which various security functions should be enforced-for example, whether a security function should be

handled at the physical hardware level, the operating system level, or the DBMS level.

1. The need in some organizations to identify multiple *security levels* and to categorize the data and users based on these classifications-for example, top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications of data must be enforced.

2.3.5 *Security threats to Database*

Threats to databases result in the loss or degradation of some or all of the following security goals. It can be generic or specific.

2.3.5.1 *Generic Threats*

1. **Loss of integrity:** Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, modification, changing the status of data, and deletion. Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decisions.
1. **Loss of availability (denial of service):** Database availability refers to making objects available to a human user or a program to which they have a legitimate right. When the database is not available it incurs a loss (otherwise life is better without the system!). So any threat that gives rise to time offline, even to check whether something has occurred, is to be avoided.
2. **Loss of confidentiality:** Database confidentiality refers to the protection of data from unauthorized disclosure. The impact of unauthorized disclosure of confidential information can range from violation of the Data Privacy Act to endanger of national security. Unauthorized, unanticipated, or unintentional

disclosure could result in loss of public confidence, embarrassment, or legal action against the organization.

2.3.5.2 SPECIFIC THREATS

1. **Commercial sensitivity:** Most financial losses through fraud arise from employees. Access controls provide both protection against criminal acts and evidence of attempts (successful or otherwise) to carry out acts detrimental to the organization, whether fraud, extraction of sensitive data or loss of availability.
2. Personal privacy and data protection: Internationally, personal data is normally subject to legislative controls. Personal data is data about an identifiable individual. Often the individual has to be alive but the method of identification is not prescribed. So a postal code for a home may in some cases identify an individual, if only one person is living at an address with the postal code. Such data needs careful handling and control. The issues are too extensive to be discussed here but the implications should be noted. Personal data needs to be identified as such. Controls must exist on the use of that data (which may restrict ad-hoc queries). Audit trails of all access and disclosure of the information need to be retained as evidence.
3. **Computer misuse:** There is also generally legislation on the misuse of computers. Misuse includes the violation of access controls and attempts to cause damage by changing the database state or introducing worms and viruses to interfere with proper operation. These offences are often extraditable. So an unauthorized access in Hong Kong using computers in France to access databases in Germany which refer to databases in America could lead to extradition to France or Germany or the USA.
4. **Audit requirements:** These are operational constraints built around the need to know who did what, who tried to do what, and where and when everything happened. They involve the detection of events (including CONNECT and GRANT transactions), providing evidence for detection, assurance as well as either defense or prosecution.

1. Counter Measures

The most resource that is commonly affected when there is a security breach is the information of the organization. The information can be financial document, security document, transaction document and so on. The following measures are adopted to mitigate threats.

1. ***Access Control:*** The security mechanism of a DBMS must include provisions for restricting access to the database as a whole; this function is called access control and is handled by creating user accounts and passwords to control login process by the DBMS.
2. ***Inference Control:*** The inference control security problem associated with databases is that of controlling the access to statistical information or summaries of values based on various criteria that can reveal the individual data. The countermeasures to statistical database security problem is called inference control measures.
3. ***Flow Control:*** Another security is that of flow control, which prevents information from flowing in such a way that it reaches unauthorized users. Channels that are pathways for information to flow implicitly in ways that violate the security policy of an organization are called covert channels**.**
4. ***Encryption:*** A final security issue is data encryption, which is used to protect sensitive data (such as credit card numbers) that is being transmitted via some type of communication network. The data is encoded using some coding algorithm. An unauthorized user who accesses encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher data.

Database security breaches can be mitigated through security mechanisms. There are two main security mechanisms and they are:

5. Discretionary access control
6. Mandatory access control

1. ***Discretionary Access Control Based On Granting and Revoking Privileges***

Discretionary Access is a way to restrict access to information based on privileges. Privileges to use a database resource are assigned and removed individually.

The typical method of enforcing discretionary access control in a database system is based on the granting and revoking of privileges. Many current relational DBMSs use some variation of this technique. The main idea is to include statements in the query language that allow the DBA and selected users to grant and revoke privileges. The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the access matrix model, where the rows of a matrix M represent *subjects* (users, accounts, programs) and the columns represent *objects* (relations, records, columns, views, operations). Each position M (i, *j)* in the matrix represents the types of privileges (read, write, update) that subject i holds on object *j*.

These privileges to users include the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).

2. ***Mandatory Access Control Based on Granting and Revoking Privileges***

The discretionary access control technique of granting and revoking privileges on relations has traditionally been the main security mechanism for relational database systems. A user either has or does not have a certain privilege. In many applications, an additional security policy is needed that classifies data and users based on security classes. This approach, known as mandatory access control, would typically be combined with the discretionary access control mechanisms. It is important to note that most commercial DBMSs currently provide mechanisms only for discretionary access control. However, the need for multilevel security exists in government, military, and intelligence applications, as well as in many industrial and corporate applications. The mandatory security mechanisms impose multilevel security and classifying data users in several

adhoc levels and then implementing appropriate security policy of the organization.

Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is the highest level and U the lowest. Other more complex security classification schemes exist, in which the security classes are organized in a lattice.

1. A subject S is not allowed read access to an object 0 unless class(S) ≥ class (O). This is known as the simple security property.
2. A subject S is not allowed to write an object 0 unless class(S) ≤ class (O). This is known as the star property (or *-property).

The first restriction is intuitive and enforces the obvious rule that no subject can read an object whose security classification is higher than the subject's security clearance. The second restriction is less intuitive. It prohibits a subject from writing an object at a lower security classification than the subject's security clearance. Violation of this rule would allow information to flow from higher to lower classifications, which violates a basic tenet of multilevel security. For example, a user (subject) with TS clearance may make a copy of an object with classification TS and then write it back as a new object with classification U, thus making it visible throughout the system. To incorporate multilevel security notions into the relational database model, it is common to consider attribute values and tuples as data objects. Hence, each attribute A is associated with a classification attribute C in the schema, and each attribute value in a tuple is associated with a corresponding security classification. In addition, in some models, a tuple classification attribute TC is added to the relation attributes to provide a classification for each tuple as a whole.

The apparent key of a multilevel relation is the set of attributes that would have formed the primary key in a regular (single-level) relation. A multilevel relation will appear to contain different data to subjects (users) with different clearance levels. In some cases, it is possible to store a single tuple in the relation at a higher classification level and produce the corresponding tuples at a lower-level classification through a process

known as ***filtering***. In other cases, it is necessary to store two or more tuples at different classification levels with the same value for the ***apparent key***. This leads to the concept of ***polyinstantiation*** where several tuples can have the same apparent key value but have different attribute values for users at different classification levels.

Comparing discretionary access control and mandatory Access Control

Discretionary Access Control (DAC) policies are characterized by a high degree of flexibility, which makes them suitable for a large variety of application domains. The main drawback of DAC models is their vulnerability to malicious attacks, such as Trojan horses embedded in application programs. The reason is that discretionary authorization models do not impose any control on how information is propagated and used once it has been accessed by users authorized to do so. By contrast, mandatory policies ensure a high degree of protection-in a way, they prevent any illegal flow of information. They are therefore suitable for military types of applications, which require a high degree of protection. However, mandatory policies have the drawback of being too rigid in that they require a strict classification of subjects and objects into security levels, and therefore they are applicable to very few environments. In many practical situations, discretionary policies are preferred because they offer a better trade-off between security and applicability.

2.3.6.2 Security Features to Control Users Access To Data.

1. **User Identification/Authentication:**

   Users accessing a database must first connect to the DBMS and then establish a session with the database server. Some systems, for example Oracle, Sybase, and SQLBase, provide a complete identification iend authentication mechanism requiring every user connecting to the database to first identify himself to the database server with some assigned user identifier and then provide his password for authentication. Other DBMSs, such as Ingres and Informix, leave the authentication task to the operating system.

1. **System Privileges:**

   System privileges allow the execution of database operations. Examples of system privileges are: create, alter, and drop objects (tables, views, and procedures); create, alter, and drop subjects (users and groups); and start up and shut down the database server. Generally, DBMSs reserve the privileges to start up and shut down the database server and to create, alter, and drop subjects to a special user called a Database Administrator (DBA). The DBA can grant other users' privileges to create, modify, and drop tables.

1. **Access Privileges:**

   Most DBMSs allow specification of authorizations stating the types of accesses each subject is allowed to exercise on the tables. Types of accesses for which authorizations can be specified include select, insert, update, and delete. Some systems, such as Oracle, Sybase, and Informix, allow access authorizations to be specified for single columns in the tables. Others, such as Ingres and SQLbase, allow column-level authorizations only for the update access mode. Some systems (e.g., Oracle and Informix) also permit authorizations for the reference access mode, which allow users to refer to a table as a parent of a referential integrity constraint. Moreover, some systems permit authorizations for the execute access mode on procedures, which allows users to execute canned programs.

2. Authorization Administration:

Authorization administration regulates who can give authorizations to subjects. Almost all DBMSs support the role of DBA. The DBA is a privileged user who can execute system privileged operations. The DBA can also grant subjects (users and groups) system and access privileges. Although the DBA is the only user who can grant system privileges, some systems also allow users to grant access privileges to other subjects based on the ownership and the grant option . Not all systems support the grant option. For example, in Ingres and SQLBase,

only the owner of a table or the DBA can grant privileges on the table to other users.

1. **Authorization Subjects:** Subjects of authorizations are generally users. Most systems, however, allow the DBA to define groups of users, in which case authorizations can be granted to groups. Authorizations given to a group are applicable to all users of the group. Almost all DBMSs have at least one group, called p & lic, that includes all users of the systems as members. In some systems, such as Ingres and Oracle, authorizations can also be specified for roles. The DBA can create roles, grant authorizations to roles, and grant roles to users. Each user is assigned a default role, which is activated when the user logs in. To use a role, a user must provide the appropriate password. Privileges needed for the execution of an application are generally granted to roles. This implies that the user must use a certain role to execute a specific application. As an example, in Ingres, the application developer associates a role with each application; in order to start the application, users must provide the role's password.

1. **Privileges :** The concept of an authorization identifier is used to refer to a user account (or group of user accounts).Informally, there are two levels for assigning privileges to use the database system:
1. ***ACCOUNT LEVEL:*** At this level, the administrator specifies the special privileges that each user, independent of the database tables (CREATE TABLE, CREATE VIEW, ALTER, MODIFY, SELECT). If a certain account does not have the CREATE TABLE privilege, no relations can be created from that account.
2. The privileges at the account level includes:
1. The CREATE SCHEMA or CREATE TABLE privilege - to create a schema or base relation.
2. The CREATE VIEW privilege.
3. The ALTER privilege - to apply schema changes such as adding or removing attributes from relations.

4. The DROP privilege - to delete relations or views.
5. The MODIFY privilege - to insert, delete, or update tuples.
6. The SELECT privilege - to retrieve information from the database by using a SELECT query.

Level of Relationship (Table Level):

At this level privileges to access every relationship or single view are controlled. Each database table is assigned an account owner, who has all privileges on that table and is responsible for granting them to other accounts. Privileges at the relation level specify for each user the individual relations on which each type of command can be applied. Some privileges also refer to individual columns (attributes) of relations.

Properties of a secured database

1. **Secrecy:** Users should not be able to see things they are not supposed to see. E.g., A student can't see other students' grades.
1. **Integrity:** Users should not be able to modify things they are not supposed to modify. Example, only instructors can assign grades.
1. **Availability**: Users should be able to see and modify things they are allowed to.

Challenges of Database Security

Considering the vast growth in volume and speed of threats to databases and information assets, research efforts need to be devoted to the following issues: data quality, intellectual property rights, and database survivability. These are only some of the main challenges that researchers in database security are trying to address.

Data Quality

The database community needs techniques and organizational solutions to assess and attest the quality of data. These techniques may include simple mechanisms such as quality stamps that are posted on Web sites. We also need techniques that provide more effective integrity semantics verification and tools for the assessment of data quality, based on techniques such as record linkage. Application-level recovery techniques are also needed for automatically repairing incorrect data. The ETL (extract, transform,

load) tools widely used to load data in data warehouses are presently grappling with these issues.

Intellectual Property Rights

With the widespread use of the Internet and intranets, legal and informational aspects of data are becoming major concerns of organizations. To address these concerns, watermarking techniques for relational data have been proposed. The main purpose of digital watermarking is to protect content from unauthorized duplication and distribution by enabling provable ownership of the content. It has traditionally relied upon the availability of a large noise domain within which the object can be altered while retaining its essential properties. However, research is needed to assess the robustness of such techniques and to investigate different approaches aimed at preventing intellectual property rights violations.

Database Survivability

Database systems need to operate and continue their functions, even with reduced capabilities, despite disruptive events such as information warfare attacks. A DBMS, in addition to making every effort to prevent an attack and detecting one in the event of occurrence, should be able to do the following:

1. **Confinement.** Take immediate action to eliminate the attacker's access to the system and to isolate or contain the problem to prevent further spread.
2. **Damage assessment.** Determine the extent of the problem, including failed functions and corrupted data.
3. **Reconfiguration.** Reconfigure to allow operation to continue in a degraded mode while recovery proceeds.
4. **Repair.** Recover corrupted or lost data and repair or reinstall failed system functions to reestablish a normal level of operation.
5. **Fault treatment.** To the extent possible, identify the weaknesses exploited in the attack and take steps to prevent a recurrence.

2.3.7 Conclusion

The various provisions a database system may make for authorization may not sufficient protection for highly sensitive data. In such cases, data may be *encrypted*. It is not possible for encrypted data to be read unless the reader knows how to decipher (*decrypt*) the encrypted data. It is difficult to ensure the privacy of individuals while allowing use of data for statistical purposes. A simple way to deal with potential security breaches is for the system to reject any query that involves fewer than some predetermined number of individuals. Another approach to security is *data pollution*. This involves the random falsification of data provided in response to a query. A similar technique involves random modification of the query itself. For both of these techniques, the goals involve a trade-off between accuracy and security. Regardless of the approach taken to security of statistical data, it is possible for a malicious user to determine individual data values. However, good techniques can make the expense in terms of cost and time sufficiently high to be a deterrent

2.3.8 Tutor Marked Assignment

1. Describe the aspects of database Survivability
2. Explain the security features of database to control users access to data
3. What is a security Threat? Explain the counter measures of security threats
4. Describe the various categories of database security threats

2.3.8 References and Further Reading

Burtescu, E. (2009). Database security–attacks and control methods. *Journal of applied quantitative methods*, *4*(4), 449-454.

Demurjian, P. S. A. (1999). *Security, authorization and authentication for entreprise computing*. CSE Technical Report TR-03-99, Dept. of Computer Science and Engineering, University of Connecticut.

Stonebraker, M., & Kemnitz, G. (1991). The POSTGRES next generation database management system. *Communications of the ACM*, *34*(10), 78-92.

Beach, B., & Platt, D. C. (2004). *U.S. Patent No. 6,728,713*. Washington, DC: U.S. Patent and Trademark Office.

Özsu, M. T., & Valduriez, P. (2011). *Principles of distributed database systems*. Springer Science & Business Media.

Ceri, S. (2017). *Distributed databases*. Tata McGraw-Hill Education.

Teorey, T. J., Lightstone, S. S., Nadeau, T., & Jagadish, H. V. (2011). *Database modeling and design: logical design*. Elsevier.

Atzeni, P., Ceri, S., Paraboschi, S., & Torlone, R. (1999). *Database systems: concepts, languages & architectures* (Vol. 1). London: McGraw-Hill.

Singh, S. K. (2011). *Database systems: Concepts, design and applications*. Pearson Education India.

Marakas, G. M., & O'Brien, J. A. (2013). *Introduction to information systems*. New York: McGraw-Hill/Irwin.

MODULE 3: DISTRIBUTED DATABASES

UNIT 1: DISTRIBUTED DATABASE SYSTEM (DDB)

3.1 Introduction:

Database is primarily used for storing and manipulating the data for the organization or any particular requirement. It contains data and its related architectures. In ideal setting, any database will have its server and more than one user. Hence when database is designed, its server is kept at one place (or computer) and users are made to access this system. A distributed database is a type of database in which storage devices are not all attached to a common central processing unit. The data may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. This unit gives an overview of distributed databases and their various forms

3.1.1 Objectives

By the end of this unit, you should be able to:

1. define Distributed database
2. know the features of distributed Databases
3. state the advantages and disadvantages of distributed databases

3.1.2 What is Distributed Database System?

A distributed database is a type of database in which storage devices are not all attached to a common central processing unit. The data may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. Collections of data can be distributed across multiple physical locations. A distributed database can reside on network servers on the internet, corporate intranets or other company networks. The replication and distribution of databases improves database performance at end-user places.

Initially when database is created, it will be like a skeleton. As and when user starts accessing the database, its size grows or shrinks. Usually the size of the database grows

drastically than shrinking. Similarly number of users may also increase. These users may not be from one single location.

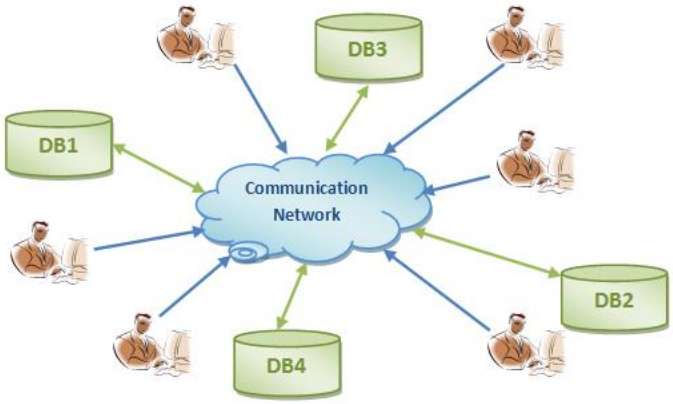


Figure 3.1: Structure of Distributed Database System

They will be from around the world. Hence the transaction with the database also increases. But this will create heavy network load as the users are at different location and server is at some other remote location. All these increasing factors affect the performance of database – it reduces its performance. But imagine systems like trading, bank accounts, etc which gives such a slow performance will lead to issues like concurrency, redundancy, security etc. Moreover users have to wait for longer time for their transaction to get executed. User cannot sit in front of their monitor to see the balance in their account for longer time.

In order to overcome these issues, a new way of allocating users and DB server is introduced. This new method is known as Distributed Database System. In this method, database server is kept at different remote locations. That means different database server is created and are placed at different locations rather than at single location. This in turn kept in sync with each other to maintain the consistency. The users accessing the DB will access any of these DB servers over the network as if they are accessing the DB from single location. They will be able to access the server without knowing its location. This in turn reduced the accessing time for the user. i.e.; when a user issues a

query, the system will fetch the server near to that user and access will be provided to nearest server. Hence it reduces the accessing time and network load too.

In ensuring that the distributive databases are up to date and current, there are two processes which are replication and duplication. Replication is using the specialized software that looks for changes in the distributive database. Once the changes have been identified, the replication process makes the entire database look the same. The replication process can be complex and also requires a lot of time with computer resources. This will depend on the size and number of the distributive database. Duplication on the other hand identifies one database as a master and then duplicates that database. A distributed database does not share main memory.

A database user accesses the distributed database through;

1. Local application which does not require data from other sites
2. Global applications which do require data from other sites.

3.1.3 Types of Distributed Database Systems

There are two types of DDB systems; Homogeneous and Heterogeneous

1. Homogeneous DDB

This type of distributed database system will have identical database systems distributed over the network. When we say identical database systems it includes software, hardware, operating systems etc – in short all the components that are essential for having a DB. For example a database system with Oracle alone distributed over the network, or with DB2 alone distributed over the network etc. this type of DDBMS system does not give the feel that they are located at different locations. Users access them as if they are accessing the same system.

Heterogeneous DDB

This is contrast to above concept. Here we will have different DBs distributed over the network. For example DB at one location can be oracle; at another location can be Sybase, DB2 or SQL server. In other words, in this type of DDB,

at least one of the DB is different from other DBs. In addition to this, the operating systems that they are using can also be different – one DB may be in Windows system while other would be in LINUX. Irrespective of the type of DDBMS used, user will be accessing these DBs as if they are accessing it locally.

3.1.4 Advantages and Disadvanatges of Distributed Databases

Although DBs helps in performance, security and recovery, there are many other advantages of this type of DBs.

1. **Transparency Levels:** In this systems, physical location of the different DBs, the data like files, tables and any other data objects are not known to the users. They will have the illusion that they are accessing the single database at one location. Thus this method gives the distribution transparency about the databases. In addition, the records of the tables can also be distributed over the databases either wholly or partially by fragmenting them. This type of system provides location transparency by allowing the user to query any database or tables from any location, network transparency by allowing to access any DB over the network, naming transparency by accessing any names of objects like tables, views etc, replication transparency by allowing to keep the copies of the records at different DBs, fragmentation transparency by allowing to divide the records in a table horizontally or vertically.

1. **Availability and Reliability:** Distribution of data among different DBs allows the user to access the data without knowing failure of any one of the system. If any system fails or crashes, data will be provided from other system. For example, if DB-IN fails, the user will be given data from DB-ALL or vice versa. User will not be given message that the data is from DB-IN or DB-ALL, i.e.; all the DBs will be in sync with each other to survive the failure. Hence data will be available to the user all the time. This will in turn guarantee the reliability of the system.

2. **Performance :** Since the users access the data that are present in the DBs which are near to them, it reduces the network load and network time. This also reduces the data management time. Hence this type of systems gives high performance.
3. **Modularity :** Suppose any new DB has to be added to this system. This will not require any complex changes to the existing system. Any new DBs can be easily added to the system, without having much change to the existing system (because the entire configuration to have multiple DB already exists in the system). Similarly, if any DB has to be modified or removed, it can also be done without much effort.

3.1.5 Disadvantages

This system also has disadvantages.

1. **Increased Complexity:** This is the main drawback of this system. Since it has many DBs, it has to maintain all of them to work together. This needs extra design and work to keep them in sync, coordinate and make them work efficiently. These extra changes to the architecture makes DDBMS complex than a DB with single server.
2. **Very Expensive:** Since the complexity is increased, cost of maintaining these complexity also increases. Cost for multiple DBs and manage them are extra compared to single DB.
3. **Difficult to maintain Integrity:** Extra effort is needed to maintain the integrity among the DBs in the network. It may need extra network resources to make it possible.
4. **Security:** Since data is distributed over the DBs and network, extra caution is to be taken to have security of data. The access levels as well unauthorized access over the network needs some extra effort.
5. DDBMS requires very good experience in DBMS to deal with.
6. **Fragmentation** of data and their distribution gives extra challenges to the developer as well as database design. This in turn increases the complexity of database design to meet the transparency, reliability, integrity and redundancy.

1. Components of Distributed Database Systems

7. Distributed Database System consists of the various components:
8. ***Database manager*** is one of major component of Distributed Database systems. He/she is responsible for handling a segment of the distributed database.
9. ***User Request Interface*** is another important component of distributed database systems. It is usually a client program which acts as an interface to the Distributed Transaction Manager.
10. ***Distributed Transaction Manager*** is a program that helps in translating the user requests and converting into format required by the database manager, which are typically distributed. A distributed database system is made of both the distributed transaction manager and the database manager.

3.1.7 Current Trends in Distributed Databases

Current trends in distributed data management are centered on the Internet, in which petabytes of data can be managed in a scalable, dynamic, and reliable fashion. Two important areas in this direction are *cloud computing and peer-to-peer databases*.

Cloud Computing

Cloud computing is the paradigm of offering computer infrastructure, platforms, and software as services over the Internet. It offers significant economic advantages by limiting both up-front capital investments toward computer infrastructure as well as total cost of ownership. It has introduced a new challenge of managing petabytes of data in a scalable fashion. Traditional database systems for managing enterprise data proved to be inadequate in handling this challenge, which has resulted in a major architectural revision.

Peer-to-Peer Database Systems

A peer-to-peer database system (PDBS) aims to integrate advantages of P2P (peerto-peer) computing, such as scalability, attack resilience, and self-

organization, with the features of decentralized data management. Nodes are autonomous and are linked only to a small number of peers individually. It is permissible for a node to behave purely as a collection of files without offering a complete set of traditional DBMS functionality. While FDBS and MDBS mandate the existence of mappings between local and global federated schemas, PDBSs attempt to avoid a global schema by providing mappings between pairs of information sources. In PDBS, each peer potentially models semantically related data in a manner different from other peers, and hence the task of constructing a central mediated schema can be very challenging. PDBSs aim to decentralize data sharing.

Problems in Distributed Database

One of the major problems in distributed systems is deadlock. A deadlock is a state where a set of processes request resources that are held by other processes in the set and none of the process can be completed. One process can request and acquire resources in any order without knowing the locks acquired by other processes. If the sequence of the allocations of resources to the processes is not controlled, deadlocks can occur. Hence we focus on deadlock detection and removal.

Deadlock Detection:

In order to detect deadlocks, in distributed systems, deadlock detection algorithm must be used. Each site maintains a local wait for graph. If there is any cycle in the graph, there is a deadlock in the system. Even though there is no cycle in the local wait for graph, there can be a deadlock. This is due to the global acquisition of resources.

In order to find the global deadlocks, global wait for graph is maintained. This is known as centralized approach for deadlock detection. The centralized approach to deadlock detection, while straightforward to implement, has two main drawbacks.

1. First, the global coordinator becomes a performance bottleneck, as well as a single point of failure.

2. Second, it is prone to detecting non-existing deadlocks, referred to as phantom deadlocks.

1. Deadlock Recovery

A deadlock always involves a cycle of alternating process and resource nodes in the resource graph. The general approach for deadlock recovery is process termination. In this method, nodes and edges of the resource graph are eliminated. In Process Termination, the simplest algorithm is to terminate all processes involved in the deadlock. This approach is unnecessarily wasteful, since, in most cases, eliminating a single process is sufficient to break the deadlock. Thus, it is better to terminate processes one at a time, release their resources, and check at each step if the deadlock still persists. Before termination of process following parameters need to be checked:

a) The priority of the process:
b) The cost of restarting the process
c) The current state of the process

3.1.8 Conclusion

A distributed database is a type of database in which storage devices are not all attached to a common central processing unit. The data may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.

3.1.9.0 Summary

In this unit we have learnt that:

i. a distributed database is a type of database in which storage devices are not all attached to a common central processing unit. The data may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.

ii) The two types of DDB are Homogeneous and heterogeneous systems

iii) Advantages and disadvantages of DDBMS

3.1.10 Tutor Marked Assignment

1. Define DDBs and state is advantages and disadvantages
2. Give suitable structure of a distributed database System
3. Enumerate the challenges of distributed database systems

1. Further Reading and other Resources

1. **David M. Kroenke, David J. Auer** (2008). Database Concepts. New Jersey . Prentice Hall
2. **Elmasri Navathe** (2003). Fundamentals of Database Systems. England. Addison Wesley.
3. **Fred R. McFadden, Jeffrey A. Hoffer** (1994). Modern Database management. England., Addison Wesley Longman

MODULE 3: DISTRIBUTED DATABASES

UNIT 2: ENHANCED DATABASE MODELS

3.2.1 Introduction:

This unit provides an overview of the network data model and hierarchical data model. The original network model and language were presented in the CODASYL Data Base Task Group's 1971 report; hence it is sometimes called the DBTG model. Revised reports in 1978 and 1981 incorporated more recent concepts. In this unit, rather than concentrating on the details of a particular CODASYL report, we present the general concepts behind network-type databases and use the term **network model** rather than CODASYL model or DBTG model. The original CODASYL/DBTG report used COBOL as the host language. Regardless of the host programming language, the basic database manipulation commands of the network model remain the same. Although the network model and the object-oriented data model are both navigational in nature, the data structuring capability of the network model is much more elaborate and allows for explicit insertion/deletion/modification semantic specification. However, it lacks some of the desirable features of the object models.

There are no original documents that describe the hierarchical model, as there are for the relational and network models. The principles behind the hierarchical model are derived from Information Management System (IMS), which is the dominant hierarchical system in use today by a large number of banks, insurance companies, and hospitals as well as several government agencies.

3.2.1 Concepts of Network Data Modeling

There are two basic data structures in the network model: records and sets.

3.2.1.1 Records, Record Types, and Data Items

Data is stored in **records;** each record consists of a group of related data values. Records are classified into **record types,** where each record type describes the structure of a group of records that store the same type of information. Each record type is given a name, and format (data type) for each **data item** (or attribute) in the record type. Figure 3.2.1 shows a record type STUDENT with data items NAME, SSN, ADDRESS, MAJORDEPT, and BIRTHDATE.

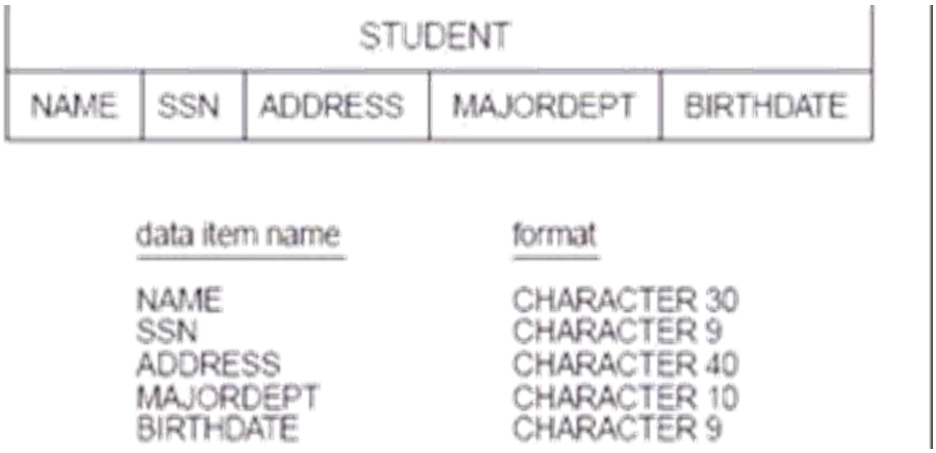


Figure 3.2.1 Illustration of record type STUDENT with data items

We can declare a virtual data item (or derived attribute) AGE for the record type shown in Figure above and write a procedure to calculate the value of AGE from the value of the actual data item BIRTHDATE in each record. A typical database application has numerous record types—from a few to a few hundred. To represent relationships between records, the network model provides the modeling construct called *set type*, which we discuss next.

Set Types and Their Basic Properties

A **set type** is a description of a 1:N relationship between two record types. Figure 3.2.2 shows how we represent a set type diagrammatically as an arrow. This type of diagrammatic representation is called a **Bachman diagram.** Each set type definition consists of three basic elements:

1. A name for the set type.

2. An owner record type.
3. A member record type.

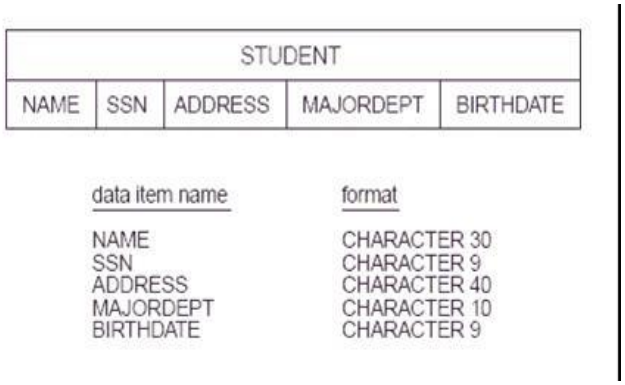


Figure 3.2.2. Representation of a set type

The set type in Figure 3.2.2 is called MAJOR_DEPT; DEPARTMENT is the **owner** record type, and STUDENT is the **member** record type. This represents the 1:N relationship between academic departments and students majoring in those departments. In the database itself, there will be many **set occurrences** (or **set instances**) corresponding to a set type. Each instance relates one record from the owner record type—a DEPARTMENT record in our example—to the set of records from the member record type related to it—the set of STUDENT records for students who major in that department. Hence, each set occurrence is composed of:

1. One owner record from the owner record type.
2. A number of related member records (zero or more) from the member record type.

A record from the member record type *cannot exist in more than one set occurrence* of a particular set type. This maintains the constraint that a set type represents a 1:N relationship. In our example a STUDENT record can be related to at most one major DEPARTMENT and hence is a member of at most one set occurrence of the MAJOR_DEPT set type.

A set occurrence can be identified either by the *owner record* or by *any of the member records.* Figure 3.2.3, shows four set occurrences (instances) of the MAJOR_DEPT set

type. Notice that each set instance *must* have one owner record but can have any number of member records (zero or more). Hence, we usually refer to a set instance by its owner record. The four set instances in Figure 3.2.3 can be referred to as the 'Computer Science', 'Mathematics', 'Physics', and 'Geology' sets. It is customary to use a different representation of a set instance where the records of the set instance are shown linked together by pointers, which corresponds to a commonly used technique for implementing sets.

In the network model, a set instance is *not identical* to the concept of a set in mathematics. There are two principal differences:

1. The set instance has one *distinguished element*—the owner record—whereas in a mathematical set there is no such distinction among the elements of a set.
1. In the network model, the member records of a set instance are *ordered,* whereas order of elements is immaterial in a mathematical set. Hence, we can refer to the first, second, i^{th}, and last member records in a set instance. Figure 3.2.4 shows an alternate "linked" representation of an instance of the set MAJOR_DEPT.

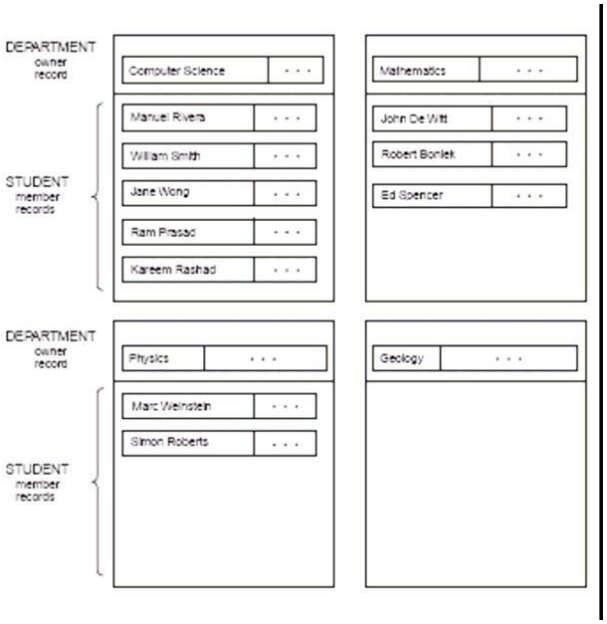


Figure 3.2.3 Four set instances of the set type MAJOR_DEPT.

In Figure 3.2.4, the record of 'Manuel Rivera' is the first STUDENT (member) record in the 'Computer Science' set, and that of 'Kareem Rashad' is the last member record. The set of the network model is sometimes referred to as an **owner-coupled set** or **co-set,** to distinguish it from a mathematical set.

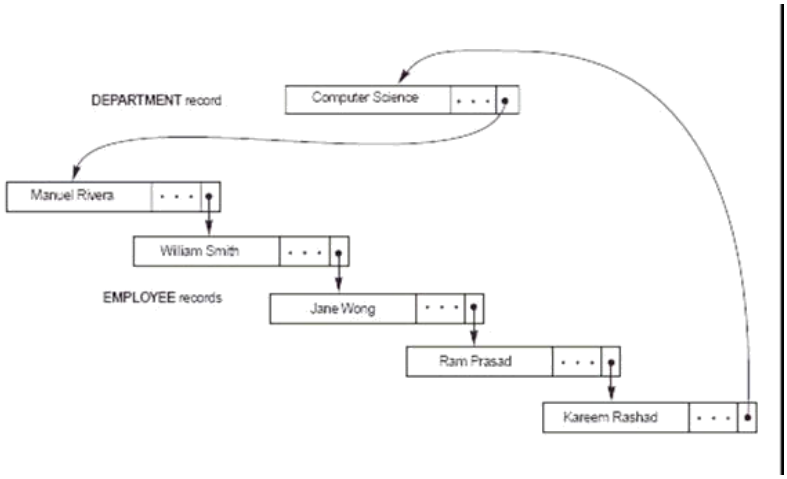


Figure 3.2.4: Alternate representation of a set instance as a linked list.

1. Special Types of Sets

System-owned (Singular) Sets. One special type of set in the CODASYL network model is worth mentioning: SYSTEM-owned sets.

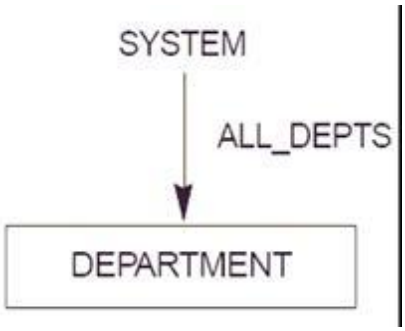


Figure 3.2.5: A singular (SYSTEM-owned) set ALL_DEPTS.

A **system-owned** set is a set with no owner record type; instead, the system is the owner. We can think of the system as a special "virtual" owner record type with only

a single record occurrence. System-owned sets serve two main purposes in the network model:

1. They provide *entry points* into the database via the records of the specified member record type. Processing can commence by accessing members of that record type, and then retrieving related records via other sets.
2. They can be used to *order* the records of a given record type by using the set ordering specifications. By specifying several system-owned sets on the same record type, a user can access its records in different orders.

A system-owned set allows the processing of records of a record type by using the regular set operations. This type of set is called a singular set because there is only one set occurrence of it. The diagrammatic representation of the system-owned set ALL_DEPTS is shown in Figure 3.2.5, which allows DEPARTMENT records to be accessed in order of some field—say, NAME—with an appropriate set-ordering specification. Other special set types include recursive set types, with the same record serving as an owner and a member, which are mostly disallowed; multimember sets containing multiple record types as members in the same set type are allowed in some systems.

3.2.2 Stored Representations of Set Instances

A set instance is commonly represented as a **ring (circular linked list)** linking the owner record and all member records of the set, as shown in Figure 10.4. This is also sometimes called a **circular chain.** The ring representation is symmetric with respect to all records; hence, to distinguish between the owner record and the member records, the DBMS includes a special field, called the **type field that** has a distinct value (assigned by the DBMS) for each record type. By examining the type field, the system can tell whether the record is the owner of the set instance or is one of the member records. This type field is hidden from the user and is used only by the DBMS.

In addition to the type field, a record type is automatically assigned a **pointer field** by the DBMS for *each set type in which it participates as owner or member.* This pointer can be considered to be *labeled* with the set type name to which it corresponds; hence, the system internally maintains the correspondence between these pointer fields and their set types. A pointer is usually called the **NEXT pointer** in a member record and the **FIRST pointer** in an owner record because these point to the next and first member records, respectively. In our example of Figure 3.2.3, each student record has a NEXT pointer to the next student record within the set occurrence. The NEXT pointer of the *last member record* in a set occurrence points back to the owner record. If a record of the member record type does not participate in any set instance, its NEXT pointer has a special **nil** pointer. If a set occurrence has an owner but no member records, the FIRST pointer points right back to the owner record itself or it can be **nil.**

The preceding representation of sets is one method for implementing set instances. In general, a DBMS can implement sets in various ways. However, the chosen representation must allow the DBMS to do all the following operations:

1. Given an owner record, find all member records of the set occurrence.
2. Given an owner record, find the first, i^{th}, or last member record of the set occurrence. If no such record exists, return an exception code.
3. Given a member record, find the next (or previous) member record of the set occurrence. If no such record exists, return an exception code.
4. Given a member record, find the owner record of the set occurrence.

The circular linked list representation allows the system to do all of the preceding operations with varying degrees of efficiency. In general, a network database schema has many record types and set types, which means that a record type may participate as owner and member in numerous set types. For example, in the network schema that appears later as Figure 3.2.7, the EMPLOYEE record type participates as owner in four set TYPES—MANAGES, IS_A_SUPERVISOR, E_WORKSON, and

DEPENDENTS_OF—and participates as member in two set types—WORKS_FOR and SUPERVISEES. In the circular linked list representation, six additional pointer fields are added to the EMPLOYEE record type. However, no confusion arises, because each pointer is labeled by the system and plays the role of FIRST or NEXT pointer for a *specific set type.*

3.2.2.1 Using Sets to Represent M:N Relationships

A set type represents a 1:N relationship between two record types. This means that *a record of the member record type can appear in only one set occurrence.* This constraint is automatically enforced by the DBMS in the network model. To represent a 1:1 relationship, the extra 1:1 constraint must be imposed by the application program.

An M:N relationship between two record types cannot be represented by a single set type. For example, consider the WORKS_ON relationship between EMPLOYEEs and

PROJECTs. Assume that an employee can be working on several projects simultaneously and that a project typically has several employees working on it. If we try to represent this by a set type, neither the set type in Figure 3.2.6a nor that in Figure 3.2.6 (b) will represent the relationship correctly. Figure 3.2.6 (a) enforces the incorrect constraint that a PROJECT record is related to only one EMPLOYEE record, whereas Figure 3.2.6 (b) enforces the incorrect constraint that an EMPLOYEE record is related to only one PROJECT record. Using both set types E_P and P_E simultaneously, as in Figure 3.2.6 (c), leads to the problem of enforcing the constraint that P_E and E_P are mutually consistent inverses, plus the problem of dealing with relationship attributes.

The correct method for representing an M:N relationship in the network model is to use two set types and an additional record type, as shown in Figure 3.2.6(d). This additional record type—WORKS_ON, in our example—is called a **linking** (or

dummy) record type. Each record of the WORKS_ON record type must be owned by one EMPLOYEE record through the E_W set and by one PROJECT record through the P_W set and serves to relate these two owner records.

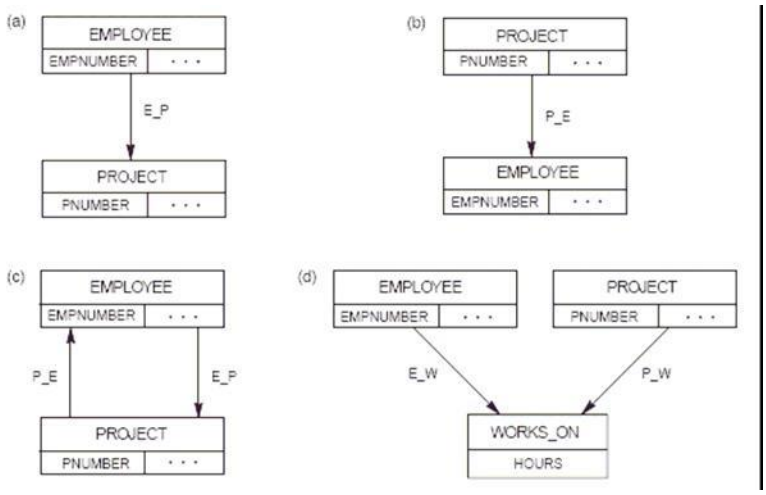


Figure 3.2.6: Representing M:N relationships. (a)–(c) Incorrect representations. (d) Correct representation using a linking record type.

Figure 3.2.6 shows an example of individual record and set occurrences in the linked list representation corresponding to the schema in Figure 3.2.6(d). Each record of the WORKS_ON record type has two NEXT pointers: the one marked NEXT(E_W) points to the next record in an instance of the E_W set, and the one marked NEXT(P_W) points to the next record in an instance of the P_W set. Each WORKS_ON record relates its two owner records. Each WORKS_ON record also contains the number of hours per week that an employee works on a project. The same occurrences in Figure 3.2.6 (f) are shown in Figure 3.2.6 (e) by displaying the W records individually, without showing the pointers.

To find all projects that a particular employee works on, we start at the EMPLOYEE record and then trace through all WORKS_ON records owned by that EMPLOYEE, using the FIRST(E_W) and NEXT(E_W) pointers. At each WORKS_ON record in the set occurrence, we find its owner PROJECT record by following the NEXT(P_W) pointers until we find a record of type PROJECT. For example, for the

E2 EMPLOYEE record, we follow the FIRST (E_W) pointer in E2 leading to W1, the NEXT(E_W) pointer in W1 leading to W2, and the NEXT(E_W) pointer in W2 leading back to E2. Hence, W1 and W2 are identified as the member records in the set occurrence of E_W owned by E2. By following the NEXT(P_W) pointer in W1, we reach P1 as its owner; and by following the NEXT(P_W) pointer in W2 (and through W3 and W4), we reach P2 as its owner. Notice that the existence of direct OWNER pointers for the P_W set in the WORKS_ON records would have simplified the process of identifying the owner PROJECT record of each WORKS_ON record.

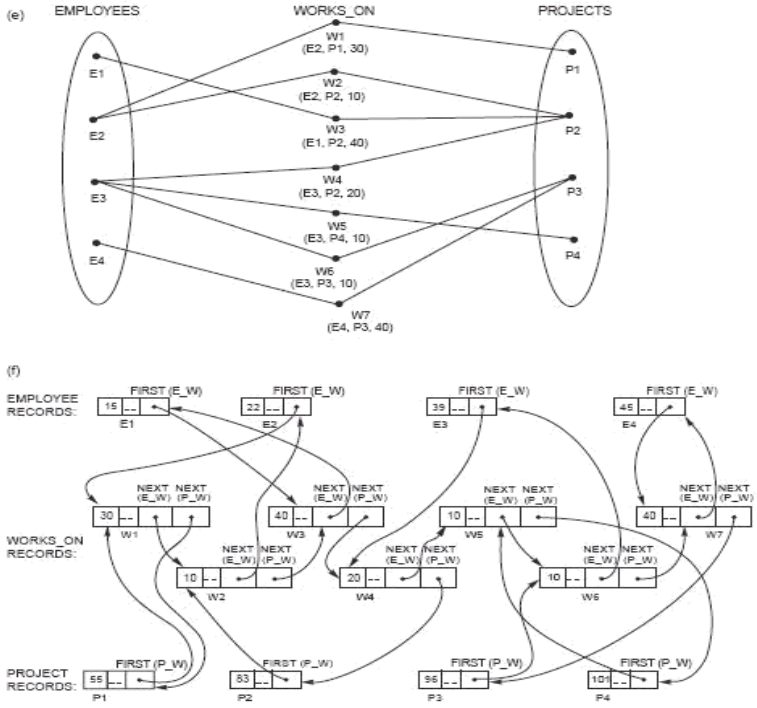


Figure 3.2.6. *(Continued)* (e) Some instances. (f) Using linked representation.

In a similar fashion, we can find all EMPLOYEE records related to a particular PROJECT. In this case the existence of owner pointers for the E_W set would simplify processing. All this pointer tracing is done *automatically by the DBMS;* the programmer has DML commands for directly finding the owner or the next member.

Notice that we could represent the M:N relationship as in Figure 3.2.6(a) or Figure 3.2.6(b) if duplicate PROJECTs (or EMPLOYEEs) records are allowed. In Figure 3.2.6a PROJECT record would be duplicated as many times as there were employees working on the project. However, duplicating records creates problems in maintaining consistency among the duplicates whenever the database is updated, and it is not recommended in general.

3.2.3 Constraints in the Network Model

In explaining the network model so far, we have already discussed "structural" constraints that govern how record types and set types are structured. In the present unit we will discuss "behavioral" constraints that apply to (the behavior of) the members of sets when insertion, deletion, and update operations are performed on sets. Several constraints may be specified on set membership. These are usually divided into two main categories, called **insertion options** and **retention options** in CODASYL terminology. These constraints are determined during database design by knowing how a set is required to behave when member records are inserted or when owner or member records are deleted. The constraints are specified to the DBMS when we declare the database structure, using the data definition language. Not all combinations of the constraints are possible. We first discuss each type of constraint and then give the allowable combinations.

1. Insertion Options (Constraints) on Sets

The insertion constraints—or options, in CODASYL terminology—on set membership specify what is to happen when we insert a new record in the database that is of a member record type. A record is inserted by using the STORE command. There are two options:

1. **Automatic:** The new member record is *automatically connected* to an appropriate set occurrence when the record is inserted.
2. **Manual:** The new record is not connected to any set occurrence. If desired, the programmer can explicitly (*manually*) connect the record to a set occurrence subsequently by using the CONNECT command.

For example, consider the MAJOR_DEPT set type of Figure 2.3.7. In this situation we can have a STUDENT record that is not related to any department through the MAJOR_DEPT set (if the corresponding student has not declared a major). We should therefore declare the MANUAL insertion option, meaning that when a member STUDENT record is inserted in the database it is not automatically related to a DEPARTMENT record through the MAJOR_DEPT set. The database user may later insert the record "manually" into a set instance when the corresponding student declares a major department. This manual insertion is accomplished by using an update operation called CONNECT, submitted to the database system.

The AUTOMATIC option for set insertion is used in situations where we want to insert a member record into a set instance automatically upon storage of that record in the database. We must specify a criterion for *designating the set instance* of which each new record becomes a member. As an example, consider the set type shown in Figure 3.2.7(a), which relates each employee to the set of dependents of that employee. We can declare the EMP_DEPENDENTS set type to be AUTOMATIC, with the condition that a new DEPENDENT record with a particular EMPSSN value is inserted into the set instance owned by the EMPLOYEE record with the same SSN value.

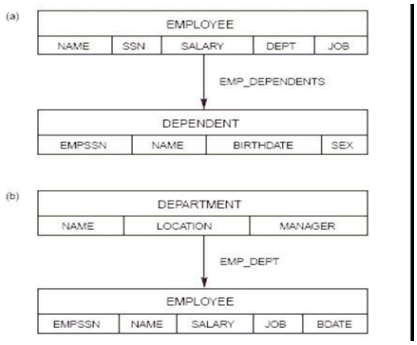


Figure 3.2.7: different set options. (a) An AUTOMATIC FIXED set.
(b) An AUTOMATIC MANDATORY set.

3.2.3.2 Retention Options (Constraints) on Sets

The retention constraints—or options, in CODASYL terminology—specify whether a record of a member record type can exist in the database on its own or whether it must always be related to an owner as a member of some set instance. There are three retention options:

1. **OPTIONAL:** A member record can exist on its own *without being* a member in any occurrence of the set. It can be connected and disconnected to set occurrences at will by means of the CONNECT and DISCONNECT commands of the network DML.
2. **MANDATORY:** A member record *cannot* exist on its own; it must *always* be a member in some set occurrence of the set type. It can be reconnected in a single operation from one set occurrence to another by means of the RECONNECT command of the network DML.
3. **FIXED:** As in MANDATORY, a member record *cannot*exist on its own. Moreover, once it is inserted in a set occurrence, it is *fixed;* it *cannot* be reconnected to another set occurrence.

We now illustrate the differences among these options by examples showing when each option should be used. First, consider the MAJOR_DEPT set type of Figure 3.2.7. To provide for the situation where we may have a STUDENT record that is not related to any department through the MAJOR_DEPT set, we declare the set to be OPTIONAL. In Figure 11(a) EMP_DEPENDENTS is an example of a FIXED set type, because we do not expect a dependent to be moved from one employee to another. In addition, every DEPENDENT record must be related to some EMPLOYEE record at all times. In Figure 3.2.7(b) a MANDATORY set EMP_DEPT relates an employee to the department the employee works for. Here, every employee must be assigned to exactly one department at all times; however, an employee can be reassigned from one department to another.

By using an appropriate insertion/retention option, the DBA is able to specify the behavior of a set type as a constraint, which is then *automatically* held good by the system.

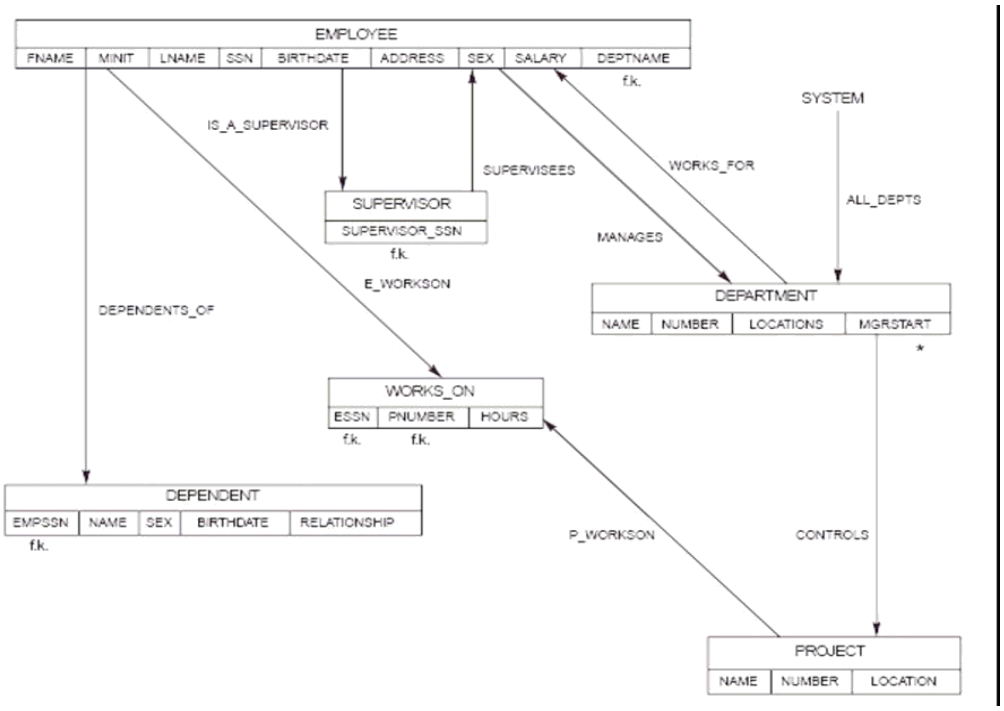


Figure 3.2.7: A network schema diagram for the COMPANY database

3.2.3.3 Data Manipulation in a Network Database

In this section we discuss how to write programs that manipulate a network database—including such tasks as searching for and retrieving records from the database; inserting, deleting, and modifying records; and connecting and disconnecting records from set occurrences. A **data manipulation language (DML)** is used for these purposes. The DML associated with the network model consists of record-at-a-time commands that are embedded in a general-purpose programming language called the **host language.** Embedded commands of the DML are also called the **data sublanguage.** In practice, the most commonly used host languages are COBOL and PL/I. In our examples, however, we show program segments in PASCAL notation augmented with network DML commands.

3.2.4 Basic Concepts for Network Database Manipulation

To write programs for manipulating a network database, we first need to discuss some basic concepts related to how data manipulation programs are written. The database system and the host programming language are two separate software systems that are linked together by a common interface and communicate only through this interface. Because DML commands are record-at-a-time, it is necessary to identify specific records of the database as **current records.** The DBMS itself keeps track of a number of current records and set occurrences by means of a mechanism known as **currency indicators.** In addition, the host programming language needs local program variables to hold the records of different record types so that their contents can be manipulated by the host program. The set of these local variables in the program is usually referred to as the **user work area (UWA).** The UWA is a set of program variables, declared in the host program, to communicate the contents of individual records between the DBMS and the host program. For each record type in the database schema, a corresponding program variable with the same format must be declared in the program.

Currency Indicators

In the network DML, retrievals and updates are handled by moving or **navigating** through the database records; hence, keeping a trace of the search is critical. Currency indicators are a means of keeping track of the most recently accessed records and set occurrences by the DBMS. They play the role of position holders so that we may process new records starting from the ones most recently accessed until we retrieve all the records that contain the information we need. Each currency indicator can be thought of as a record pointer (or record address) that points to a single database record. In a network DBMS, several currency indicators are used:

1. ***Current of record type****:* For each record type, the DBMS keeps track of the most recently accessed record of that record type. If no record has been accessed yet from that record type, the current record is undefined.

2. ***Current of set type:*** For each set type in the schema, the DBMS keeps track of the most recently accessed set occurrence from the set type. The set occurrence is specified by a single record from that set, which is either the owner or one of the member records. Hence, the current of set (or current set) points to a record, even though it is used to keep track of a set occurrence. If the program has not accessed any record from that set type, the current of set is undefined.
3. ***Current of run unit (CRU):*** A run unit is a database access program that is executing (running) on the computer system. For each run unit, the CRU keeps track of the record most recently accessed by the program; this record can be from *any* record type in the database.

Each time a program executes a DML command, the currency indicators for the record types and set types affected by that command are updated by the DBMS.

Status Indicators

Several **status indicators** return an indication of success or failure after each DML command is executed. The program can check the values of these status indicators and take appropriate action—either to continue execution or to transfer to an error-handling routine. We call the main status variable DB_STATUS and assume that it is implicitly declared in the host program. After each DML command, the value of DB_STATUS indicates whether the command was successful or whether an error or an exception occurred. The most common exception that occurs is the **END_OF_SET (EOS)** exception.

3.2.5 Hierarchical Model

3.2.5.1 Parent-Child Relationships and Hierarchical Schemas

The hierarchical model employs two main data structuring concepts: records and parent-child relationships. A **record** is a collection of **field values** that provide information on an entity or a relationship instance. Records of the same type are grouped into **record types.** A record type is given a name, and its structure is defined by a collection of

named **fields** or **data items.** Each field has a certain data type, such as integer, real, or string.

A **parent-child relationship type (PCR type)** is a 1:N relationship between two record types. The record type on the 1-side is called the **parent record type,** and the one on the N-side is called the **child record type** of the PCR type. An **occurrence** (or **instance**) of the PCR type consists of *one record* of the parent record type and a number of records (zero or more) of the child record type.

A **hierarchical database schema** consists of a number of hierarchical schemas. Each **hierarchical schema** (or **hierarchy**) consists of a number of record types and PCR types. A hierarchical schema is displayed as a **hierarchical diagram,** in which record type names are displayed in rectangular boxes and PCR types are displayed as lines connecting the parent record type to the child record type. Figure 10.9 shows a simple hierarchical diagram for a hierarchical schema with three record types and two PCR types. The record types are DEPARTMENT, EMPLOYEE, and PROJECT. Field names can be displayed under each record type name as shown in Figure 3.2.8. In some diagrams, for brevity, we display only the record type names.

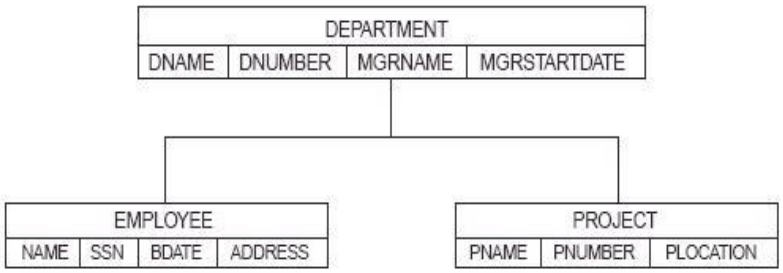


Figure 3.2.8**: A hierarchical schema.**

We refer to a PCR type in a hierarchical schema by listing the pair (parent record type, child record type) between parentheses. The two PCR types in Figure 3.2.8 are (DEPARTMENT, EMPLOYEE) and (DEPARTMENT, PROJECT). Notice that PCR types *do not* have a name in the hierarchical model. In Figure 3.2.8 each *occurrence* of

the (DEPARTMENT, EMPLOYEE) PCR type relates one department record to the records of the *many* (zero or more) employees who work in that department. An *occurrence* of the (DEPARTMENT, PROJECT) PCR type relates a department record to the records of projects controlled by that department. Figure 3.2.9 shows two PCR occurrences (or instances) for each of these two PCR types.

Properties of a Hierarchical Schema

A hierarchical schema of record types and PCR types must have the following properties:

1. One record type, called the **root** of the hierarchical schema, does not participate as a child record type in any PCR type.
2. Every record type except the root participates as a child record type in *exactly one* PCR type.
3. A record type can participate as parent record type in any number (zero or more) of PCR types.
4. A record type that does not participate as parent record type in any PCR type is called a **leaf** of the hierarchical schema.
5. If a record type participates as parent in more than one PCR type, then *its child record types are ordered.* The order is displayed, by convention, from left to right in a hierarchical diagram.

The definition of a hierarchical schema defines a **tree data structure.** In the terminology of tree data structures, a record type corresponds to a **node** of the tree, and a PCR type corresponds to an **edge** (or **arc**) of the tree. We use the terms *node* and *record type*, and *edge* and *PCR type*, interchangeably. The usual convention of displaying a tree is slightly different from that used in hierarchical diagrams, in that each tree edge is shown separately from the other edge. The preceding properties of a hierarchical schema mean that every node except the root has exactly one parent node. However, a node can have several child nodes, and in this case they are ordered from left to right. In Figure 3.2.9, EMPLOYEE is the first child of DEPARTMENT, and PROJECT is the second child. The previously identified properties also limit the types of relationships that can be

represented in a hierarchical schema. In particular, M:N relationships between record types *cannot* be directly represented, because parent-child relationships are 1:N relationships, and a record type *cannot participate as child* in two or more distinct parent-child relationships. An M:N relationship may be handled in the hierarchical model by allowing duplication of child record instances. For example, consider an M:N relationship between EMPLOYEE and PROJECT, where a project can have several employees working on it, and an employee can work on several projects. We can represent the relationship as a (PROJECT, EMPLOYEE) PCR type. In this case a record describing the same employee can be duplicated by appearing once under *each* project that the employee works for. Alternatively, we can represent the relationship as an (EMPLOYEE, PROJECT) PCR type, in which case project records may be duplicated

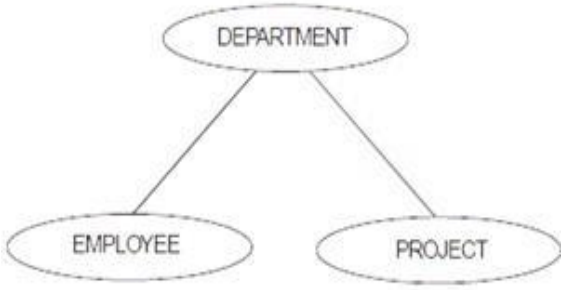


Figure 3.2.9: A tree representation of the hierarchical schema

Project	Employees Working on the Project
1.	E1, E3, E5
2.	E2, E4, E6
3.	E1, E4
4.	E2, E3, E4, E5

If these instances are stored using the hierarchical schema (PROJECT, EMPLOYEE) (with PROJECT as the parent), there will be four occurrences of the (PROJECT, EMPLOYEE) PCR type—one for each project. The employee records for E1, E2, E3, and E5 will appear *twice each* as child records, however, because each of these employees works on two projects. The employee record for E4 will appear three times—once under each of projects B, C, and D and may have number of hours that E4 works

on each project in the corresponding instance. To avoid such duplication, a technique is used whereby several hierarchical schemas can be specified in the same hierarchical database schema. Relationships like the preceding PCR type can now be defined across different hierarchical schemas. This technique, called **virtual relationships,** causes a departure from the "strict" hierarchical model.

3.2.5.1 Hierarchical Model Integrity Constraints in the Hierarchical Model

A number of built-in **inherent constraints** exist in the hierarchical model whenever we specify a hierarchical schema. These include the following constraints:

1. No record occurrences except root records can exist without being related to a parent record occurrence. This has the following implications:
   a. A child record cannot be inserted unless it is linked to a parent record.
   b. A child record may be deleted independently of its parent; however, deletion of a parent record automatically results in deletion of all its child and descendent records.
   c. The above rules do not apply to virtual child records and virtual parent records.
2. If a child record has two or more parent records from the *same* record type, the child record must be duplicated once under each parent record.

1. A child record having two or more parent records of *different* record types can do so only by having at most one real parent, with all the others represented as virtual parents. IMS limits the number of virtual parents to one.

2. In IMS, a record type can be the virtual parent in *only one* VPCR type. That is, the number of virtual children can be only one per record type in IMS.

3.2.6 Conclusion

The original network model and language were presented in the CODASYL Data Base Task Group's 1971 report; hence it is sometimes called the DBTG model. Revised reports in 1978 and 1981 incorporated more recent concepts. In this unit, rather than

concentrating on the details of a particular CODASYL report, we present the general concepts behind network-type databases and use the term **network model** rather than CODASYL model or DBTG model. There are no original documents that describe the hierarchical model, as there are for the relational and network models. The principles behind the hierarchical model are derived from Information Management System (IMS), which is the dominant hierarchical system in use today by a large number of banks, insurance companies, and hospitals as well as several government agencies.

3.2.7 Tutor Marked Assignment

1. Differentiate between status and currency indicators?
2. Mention and discuss the integrity constraints of hierarchical and network database models
3. Explain the Network model and state the constraints associated with Network model.

3.2.8 References/Suggested Readings

1. Date, C.J., Introduction to Database Systems (7 Edition) Addison Wesley, 2000
2. Leon, Alexis and Leon, Mathews, Database Management Systems, LeonTECHWorld rd
3. Elamasri R. and Navathe, S., Fundamentals of Database Systems (3 Edition), Pearsson Education, 2000.

MODULE 3: DISTRIBUTED DATABASES

UNIT 3: OBJECT ORIENTED DATABASE

3.3.0 INTRODUCTION

History of data processing goes through many different changes with different technologies along with the time. In decade there is huge increase in the volume of data that need to be processed due to which sometimes old technology do not work and need to come with new technology to process the data. History of database technology has used Unit Records and Punch Card, Punch Card Proliferation, Paper Data Reels, and Data Drums, File Systems, Database Systems, NoSQL and NewSQL databases. From last five decades, the mostly used technology is database management systems.

After some limitations of file systems, researchers come up with new technology known as Database Management Systems which is the collection of software or programs to maintain the data records. Initially, two models are proposed are hierarchical and network models, but these models don't get much popularity due to their complex nature. Then a researcher E.F. Codd comes up with a new data model known as relational model in which data items are stored in a table. Many DBMS's are developed on the basis of this model. This is the most popular model till now because it has conceptually foundation from relational mathematics.

In mid-1980's, no doubt RDBMS are very much popular but due to some limitation of relation model and RDBMS do not support for some advanced Applications. Object Oriented Database (OODB) comes in the picture. At that time Object Oriented Programming paradigm is very much popular. Due to this researcher think to combine the capabilities of database and object based paradigm for programming. In Object databases data is stored in the forms of objects. These database management systems are not very much popular because due to the lack of standards.

The term "object-oriented database system" was first introduced in 1985. Object-oriented databases are designed and built according to the object-oriented paradigm in which everything is modeled as objects including the data. This type of data model helps in tackling complex data structures, for instance multimedia content, in a more natural way and provides a seamless transition from design to conception. . Hence the data in OODBMS is represented as collection of interacting objects instead of collection of inter-related tables. Usage of object-oriented concepts like polymorphism and inheritance make the interaction between the objects a trivial task. Whereas data is stored as tables in the relational database and we need to relate of "join" tables to perform a query, *it is stored as a collection of objects in object-oriented database,* and query can be easily performed by following the pointer from parent object to its children.

3.3.1 Objectives

At the end of this unit, students should be able to:

3. Understand Object Oriented Database.
4. Understand the various models of OO database and be able to implement them.
5. Understand and state the advantages and disadvantages of OO database.

3.2. *What is Object Oriented Database?*

Object Oriented Database is a database in which information is represented in the form of objects as used in object-oriented programming. OODs are different from relational databases which are table-oriented. The requirements for a database to be Object Oriented include:

1. **It should be a Database Management System (DBMS):** This means that the OODBMS should provide the basic features for any database system – *persistence, concurrency, data recovery*, secondary storage management and ad hoc query facility.
2. It should support Polymorphism and Inheritance: This means that the database system should support all the requisite features of an object-oriented system like *encapsulation, complex objects, inheritance, polymorphism, extensibility*.

1. Basic Concepts of OO Programming

1. Object and Class: A conceptual **entity** is anything that exists and can be distinctly identified. Examples, a person, an employee, a car, a part**.** In an OO system, all conceptual entities are modeled as **objects**. An object has **structural** properties defined by a finite set of attributes and behavioural properties. The only between entity and object is that entity has only state has no behaviour, but object contains both state and behaviour.

Each object is associated with a logical non-reusable and unique object identifier (**OID**). The OID of an object is independent of the values of its attributes. All objects with the same set of attributes and methods are grouped into a class, and form instances of that class. OID has following characteristics:

1. It is generated by system.
2. It is unique to that object in the entire system.
3. It is used only by the system, not by the user.
4. It is independent the state of the object.

Classes are classified as lexical classes and non-lexical classes.

1. A **lexical class** contains objects that can be directly represented by their values.
2. A **non-lexical class** contains objects, each of which is represented by a set of attributes and methods. Instances of a non-lexical class are referred to by their **OID**s. Example PERSON, EMPLOYEE, PART are non-lexical classes.

1. ATTRIBUTES

The **domain** of an attribute of a non-lexical class A can be one of the following:

2. a lexical class such as integer, string. An attribute with this domain is called a **data-valued attribute**.
3. a non-lexical class **b).** An attribute with this domain is called an **entity-valued attribute**. Types of attributes include simple and complex attribute

4. **Method:** A method of an object is invoked by sending a **message** (which is normally the method name) to the object. Such a **message-passing** mechanism represents a **binary interaction** between the sender of the message and the recipient. A method's specification is represented by a **method signature**, which provides the method name and information on the types of the method's input parameters and its results. The implementation of the method is separated from the specification. This provides some degrees of **data independence.**

Methods play an important role in defining object semantics. Example, **w**hen an employee is fired, we need to delete the employee information from the employee file, delete the employee from the employee-project file, and insert the employee information into a history file. One method called "**Fire-employee**" can be defined that incorporates this sequence of actions.

5. **Abstraction:** It is process of finding important aspects of an entity and ignoring unimportant aspects such as implementation details. The properties comprise two things: State and behaviour. A **state** is models through the attributes of object and **behaviour** is models through operations executed on data by object.
6. **Encapsulation:** An object contains both current state (Attributes) and set of methods used to manipulate it. It is called encapsulation.
7. **Information Hiding***:* It is process of separates external properties of an object from its internal properties, which are hidden from external environment. These two concepts also related with abstraction.
8. **Importance:** These two concepts support the facility that internal properties of an object to be changed without affecting applications that use it, provided external properties remain same. It also provides data independence.
9. **Inheritance:** It is the special type of relationship between classes: The inheriting class inherits the some or all properties of the base class depend

which mode of inheritance is used. Special classes or inheriting classes are called subclasses and general classes are called super classes.

10. **Generalization:** It is method to create a superclass is called generalization.
11. **Specialization:** It is process of forming a sub class is called specialization.
12. **Polymorphism:** It means "many forms". It is dynamic feature which executes at run time of program. It involves the concept of overriding and overloading.
13. **Complex Objects:** An object is called complex object if it contains many sub objects and it is viewed as single object.
14. **Relationships:** It is basically an association between two things. These are represented through reference attributes, typically implemented through OID's. Types of binary relationships are:
    - One to One relationship
    - One to Many relationship
    - Many to One relationship
    - Many to Many relationship

OO Data Model Vs Hierarchical Data Model

1. The nested structure of objects and the nested structure of records in hierarchical databases are similar. The essential difference is that the OO data model uses logical and non-reusable OIDs to link related objects while the hierarchical model uses physical reusable pointers to physically link related records. Hierarchical model has no object and OID concepts.
2. OO data model allows cyclic definition within object structures. Example, a course can refer to other courses as its pre-requisite courses. To support cyclic definition in the hierarchical data model, dummy record types (e.g. prerequisite record) are needed.

1. **OO DATA MODEL VS NESTED RELATIONS:** In the nested relation approach, an attribute of a relation can itself be a relation. The nested relation

is stored physically within the base relation. This approach does not allow the nested relation to be shared among relations. There may be a redundant storage of data which can lead to updating anomalies. In the OO approach, nested relations are simulated by using the **OIDs** of tuples of a relation that are to be nested within a base relation. Because OIDs are used, sharing of tuples of nested relation is possible. There is less redundancy.

3.3.3 Why Object Oriented Databases?

There are three reasons for need of OODBMS:

1. Limitation of RDBMS
2. Need for Advanced Applications
3. Popularity of Object Oriented Paradigm

1. Limitation of RDBMS

These limitations are in relational model. Due to this, these limitations are reflected to all RDBMS. These limitations are:

1. *Poor representation of real world entities:* The Relational model cannot represent real world in proper way because it has only one semantic that is table which can represent the real world entity in proper way.
2. *Normalization is necessary, but sometimes not useful:* Normalization in RDBMS to maintain the consistency of the database, but some broken relations is not related to real world.
3. *Overloading of semantic structure:* Relational Data Model has only one semantic structure for representing data and relationship that is table. Due to this, sometimes it becomes very difficult to find out that which is going to model data or relationship?
4. *Poor support for integrity and enterprise constraints:* Constraints are very much needed for your database have to be desired data. RDM supports only limited number of constraints. The enterprise constraints are those which are defined by industry standards.

5. *Homogeneous data structure:* RDM requires homogeneous data structures like:
    - RDM assumes both horizontal and vertical homogeneity.
    - Relational mathematics algebra has only fixed number of operations due to which Relational Model operations cannot be extended.
6. ***Tables can store only atomic/single value:*** No doubt, this is property of RDM. But sometimes in many situations this property becomes its limitation.
7. ***Normalization is strongly recommended:*** Most of the situations, you have must normalize the relation make the data consistency inside your database.
8. ***Difficulty in handling recursive queries:*** There is very poor support to handle recursive queries in RDBMS. For this you must know:
    - Depth of recursive query must be known.
    - You can use the transitive closure operations to handle recursive queries in RDBMS.
9. ***Impedance mismatch:*** SQL Data Manipulation Language (DML) is lack computational completeness. To overcome this situation, you must embed the SQL with any high programming language like C++, Java, and C #. Due to there will be impedance mismatch between two language SQL and higher programming language.
10. ***Poor support for long duration transactions****:* In RDBMS, generally transactions are short lived and concurrency control techniques or mechanisms are not good for .long duration transactions.
11. ***Poor Schema Evolution support:*** Schema Evolution means making changes to schema of database at runtime without interrupt the execution of the application.

12. ***Poor Navigational Access:*** There is very poor support for the navigational access in RDBMS. There are some advanced applications need the database with deeper structural and functional foundation of capabilities that are not provided by conventional database.

B. Need for Advanced Applications

a) Computer Aided Design (CAD):

- In these types of applications, relevant data about buildings, airplanes and integrated circuit chips is stored and managed. In this type of applications, database design may be very large. Design in these types of applications is not static. This design is evolves through the times. Updates need to be propagated.
- These applications require version control and configuration management.

1. These applications require complex objects for their development. For example, a car's component may be related to other components.

- Need long duration transactions because sometimes updates are for reaching.
- Support for cooperative engineering because most of the times many people work on same design.

b) Computer Aided manufacturing (CAM):

- These application data is very much similar to CAD, but needs discrete production.
- These applications must respond to real time events.
- Generally algorithms and custom rules are used to respond to a particular situation.

c) Computer Aided Software Engineering (CASE):

- These applications manage data about the phases of software development life cycle.
- Design may be extremely large.
- Involves cooperative work.
- Need to maintain dependencies among components.
- Versioning and configuration management.

d) Network Management Systems:

- Coordinates communication services across the network.
- These systems are used for such tasks as network path management, problem management and network planning.

e) ***Other Applications:*** The Object Oriented Database also used in Office Information Systems, Multimedia systems, Digital Publishing and Geographic information Systems.

c. Popularity of Object Oriented Paradigm

Another domain that enforces the development of OODBMS is popularity of object oriented programming paradigm because a real life situation can be modelled in best way by using object oriented programming.

3.3.4 Approaches for OODBMS

a. Relational Extension Based DBMS
b. Object/Relational DBMS
c. Pure OODBMS

A. Relational Extension Based DBMS

This is the first approach that is adopted by industry and academia towards the implementations of OODBMS is to extend the relational model to provide the OO features. The advantages of this approach are:

- Stick to relational model

- Have to OO features like complex object and UDT (User Defined Types).

Design techniques for relational extensions: In mid-80's a researcher named Stonebraker in OODBMS field represent the design techniques in this field with different proposals for Extended Type System for an OODBMS should follow:

- Definition of new data types
- Definition of new operations of so defined data types.
- Implementation of access methods.
- Optimized query processing for the queries on new data types.

Other Extensions in RDBMS: The different techniques are adopted by different DBMS to support to support OO features:

- **Support for variable length "undefined" data values.** Using this support, generalized user defined data types can be represented. Like Oracle supported RAW, LONG and LONGRAW (65535 bytes). Sybase support TEXT and IMAGE up to 2GB and also others. These features were partial support for storing complex data. Such facilities were mainly used to capture non-text data like voice, medical charts and fingerprints.
- User defined procedure are associated with used defined data types.
- Example: POSTGRES

Postgres: It is developed at UC Berkeley in mid-80 by Prof. Stonebroker and his group. It is commercialized as ILLUSTRA. In this INGRES which is basically a relational database management system to support OO features. Basic idea in POSTGRES was to introduce minimum changes in the Codd's original relational model to achieve the objective. Advantage is the continuity with the previous product (INGRES) and provision of OO features in the new product.

Design objectives of POSTGRES declared by Stone braker were:

- To provide fully implemented functionality of complex objects.

- Support for User/Abstract Defined Types, operators and functions for accessing.
- To provide functionality of Active Databases and Inferencing.
- QUEL is the manipulation language in INGRES.
- POSTQUEL in POSTGRES.

B. Object/Relational DBMS

These systems have relational and object based both features by the definition. They provide similar objectives as provided by the Relational Extension approach of RDBMS. In this approach, build an object layer on the top of relational system like Open ODB and ODAPTER. They are built on different architectures like Query Server or Client/Server.

Open ODB/ODAPTER: Open ODB is an ORDBMS from HP during mid's 90 and aims to support for broad base applications. It has following features:

1. Based on Iris DBMS
2. Based on Client/Server architectures
3. Both data and applications can be shared by the user applications.
4. Clients use Application Program Interface (API) to access information.
5. OSQL is data manipulation language for Open ODB/ODAPTER.
6. Open ODB uses relational techniques to support to OO features.
7. Object Model is implemented by Object Manager. Mapping to OO schema and queries to relational ones.
8. The underlying relational storage manager is ALLBASE/SQL.

C. Pure OODBMS

These type OODB's systems are not much popular because lack of standards. There is no single definition for a single concept. For Example: An Object has

many definitions, but in RDB there is a fixed standard for or single definition for each concept like table .Here defining some definitions which are mostly accepted but not standardize.

OODB Model: It is data model that capture semantics of objects suited in object based programming paradigm. ZDONIK and MAIER give a threshold model that an Object database must have following features:

1. Database functionality like transaction management, concurrency control.
2. Facility of Object Identity (OID).
3. Facility of encapsulation.
4. Facility of complex objects.
5. Inheritance not must but may useful.

OODB: It is permanently stored and sharable collection of objects suited with an Object Data Model.

OODBMS: It is system which contains application programs which are used to manage all object oriented database activities like manipulation of objects.

The OODBMS paradigm manifesto set the minimum fundamental directional basis for an OODBMS model .These characteristics can be classified as mandatory and optional features:

OODBMS Mandatory features

1. *Support for complex objects:* A OODBMS must support for complex objects. Complex objects can be obtained by applying constructor on basic objects.
2. ***Object Identity:*** It is the unique identifier associated with every object in the system. It has following characteristics:

6. It is generated by system.
7. It is unique to that object in the entire system.

8. It is used only by the system, not by the user.
9. It is independent the state of the object.

3. ***Encapsulation:*** An OODBMS should enforce encapsulation through access objects only.
4. ***Types or Classes:*** A OODBMS must support for one of them types or classes.
5. ***Inheritance and Hierarchies:*** A OODBMS must support for concept of super classes and subclasses. The types of heritance can be: substitution, inclusion, constraint, specialization
6. ***Dynamic Binding:*** An OODBMS must support concept of dynamic binding in programming language such as overloading, overriding and late binding
7. ***Computationally Complete DML****:* To provide a support for data processing database have use computationally completely language like SQL-3.
8. ***Extensible set of data types****:* A OODBMS must support for used defined data types.
9. ***Data Persistence:*** This is basic requirement for any DBMS.A OODBMS must provide persistent by storing object in proper way.
10. ***Managing very large databases:*** A OODBMS must support for large databases. ***Concurrent Users:*** This is basic requirement for any DBMS. It must support for concurrency control.
12. ***Transaction Management:*** This is also basic requirement of any DBMS.
13. ***Query Language:*** This is also a basic requirement of any DBMS. This query language must be computationally complete.

OODBMS Optional Features

1. ***Multiple Inheritance:*** Multiple inheritance is not directly support by multiple objects oriented programming languages. An OODBMS can also support for multiple inheritance.

2. ***Type checking and inferencing:*** Type Checking and Inferencing features can be added to Object Databases.
3. ***Long duration and Nested Transactions:*** Relational database transactions are short-lived. An OODBMS can support for .long duration transactions and also for nested transactions.
4. ***Distributed databases:*** An object database may have support for distributed database which is a collection of multiple databases logically related and distributed over the network.
5. *Versions:* An OODBMS can support for version control and configuration management.

3.3.4 Achievements and Weaknesses of OODBMS

a) Achievements

1. *Support for User Defined data types:* OODBs provides the facilities of defining new user defined data types and to maintain them.
2. *OODB's allow creating new type of relationships:* OODBs allow creating a new type of relationship between objects is called inverse relationship (a binary relationship).
3. *No need of keys for identification:* Unlike, relational model, object data model uses object identity (OID) to identify object in the system.
4. *Development of Equality predicates:* In OODBs, four types equality predicates are:

1. Identity equality
2. Value equality of objects
3. Value equality of properties
4. Identity equality of properties

5. **No need of joins for OODBMS's:** OODBs has ability to reduce the need of joins.

6. ***Performance gains over RDBMS:*** Performance gains changes application to application. Applications that make the use of object identity concept having performance gains over RDBMS's.
7. *Provide Facility for versioning management:* The control mechanisms are missing in most of the RDBMS's, but it is supported by the OODBMS's.
8. ***Support for nested and long Duration transactions:*** Most of the RDBMS's do not support for long and nested transactions, but OODBMS's support for nested and long duration transactions.
9. *Development of object algebra:* Relational algebra is based on relational mathematics and fully implemented, but object algebra has not been defined in proper way. Five fundamental object preserving operators are union, difference, select, generate and map.

Weaknesses

1. *Coherency between Relational and Object Model:* Relational databases are founded in every organization. To overcome relational databases, object databases have to be providing coherent services to users to migrate from relational database to object database. Architecture of Relational model and Object model must be provide some coherency between them.
2. *Optimization of Query Expression:* Query expression optimization is done to increase the performance of the system. Optimization of queries is very important for performance gains. But due to following reasons it is difficult to optimize queries in object databases:

1. User defined data types
2. Changing variety of types
3. Complex objects, methods and encapsulation
4. Object Query language has nested structure
5. Object Identity

3. *No fixed query algebra for OODBMS's:* Due to lack of the standards in OODBMS, there is no standard query algebra for OODB. Lack of standard query algebra becomes one of the reasons for problem of query optimization. There are different query languages for different object databases.
4. *No full-fledged query system:* Query system also not fully implemented. Some query facilities lacks in Object databases like nested sub-queries, set queries and aggregation function.
5. *No facility for Views:* In relational databases, views are temporary tables. Object databases having no facility for views. An object oriented view capability is difficult to implement due to the features of Object Model such as object identity. Object oriented programming concepts like inheritance and encapsulation makes the difficult to implement views in object databases.
6. ***Security problems in Object databases:*** Security is related to authentication, authorization and accounting. Discretionary Access Control (DAC), mandatory access control (MAC) security policies are implemented in object databases to secure data. In some systems provide security on the basis of object oriented concepts like encapsulation. Object database having to facilities for authorization.
7. ***No support for schema evolution with OODBs:*** Most object databases do not allow schema evolution. Schema Evolution is facility which allows changing the schema at run time such as adding a new attributes or methods to the class, adding new superclass to the class.
8. *Consistency constraints mechanisms are not fully implemented:* Only limited numbers of features are provided by OODBMS's for uniqueness of constraints, integrity constraints and other enterprise constraints.

9. *No full-fledged facilities to implement complex objects:* No doubt, object oriented databases provide some facilities to implement the concept of complex objects. But there is no full –fledged implementation of complex objects.
10. *Interoperability between OODB and Object Oriented Systems:* In Object Oriented Programming objects are transient in nature. To provide persistent to data, OODB and OO systems need to be interoperable. Many problems may arise during interoperable between OODB and OO systems.
11. *Limited performance gains over RDBs Decrease in performance:* Performance gains changes application to application. Applications that make the use of object identity concept having performance gains over RDBMS's. But application that requires bulk database loading and does not make use of OID then performance of OODBMS is not good.
12. *Some basic features are not present:* Some basic features like triggers, meta-data management and constraints such as UNIQUE and NULL not present in object databases.

Table 3.3.1: Paramters of OODBMS and RDBMS Models

Parameter	OODBMS	RDBMS
Model	OODB Model	Relational Model
Standards	Lack of standards	Fully standardized
OO Programming	Direct and extensive	No Direct Support
Name Of Standards	ODMG-3.0	SQL2(ANSI X3H2)
Conceptually Foundation	No particular	Relational Mathematics
Relationships	Support only for simple and Complex relationships	Support only for simple relationships
User Defined Types	Full support for UDTs	Less Support for UDT's
Advanced Applications	Full support for advanced applications	No support for advanced applications
Complex Objects	Support for complex objects, but not to full extent	Less support for complex objects
Navigational Access	Good	Poor
Schema Evolution	Easy	Difficult
Transactions	Long-lived	Short-lived
Query Language	Depend upon product	SQL
Recursive Queries	Easy to handle	Difficult to handle
SQL	computationally complete	Not computationally complete
Normalization	No need	Strongly recommended
Store data in	Object	Table
Representation	Strong representation for real world entity	Poor representation for real world entity
Semantic nature	Semantically very strong	Semantically very weak
Constraints	Integrity constraints and Enterprise constraints are fully implemented	Integrity constraints and other Enterprise constraints are not fully implemented
Algebra	Object Algebra	Relational Algebra
Operations	Easily extension of operations	Fixed set of operations due to relational algebra
Multimedia data	Full support for multimedia data	Less support for multimedia data

Advantages of OODBMS

1. Enriched modeling capabilities

The object-oriented data model allows the 'real world' to be modeled more closely. The object, which encapsulates both state and behavior, is a more natural and realistic representation of real world objects. An object can store all the relationships it has with other objects, including many-to-many relationships, and objects can be formed into complex objects that the traditional data models cannot cope with easily.

2. Extensibility

OODBMSs allow new data types to be built from existing types. The ability to factor out common properties of several classes and form them into a superclass that can be shared with subclasses can greatly reduce

redundancy within system and, as we stated· at the start of this unit, is regarded as one of the main advantages of object orientation. Further, the reusability of classes promotes faster development and easier maintenance of the database and its applications.

3. **Capable of handling a large variety of data types**

   Unlike traditional databases (such as hierarchical, network or relational), the object oriented database are capable of storing different types of data, for example, pictures, voice video, including text, numbers and so on.

4. **Removal of impedance mismatch**

   A single language interface between the Data Manipulation Language (DML) and the programming language overcomes the impedance mismatch. This eliminates many of the efficiencies that occur in mapping a declarative language such as SQL to an imperative 'language such as 'C'. Most OODBMSs provide a DML that is computationally complete compared with SQL, the 'standard language of RDBMSs.

5. **More expressive query language**

   Navigational access from the object is the most common form of data access in an OODBMS. This is in contrast to the associative access of SQL (that is, declarative statements with selection based on one or more predicates). Navigational access is more suitable for handling parts explosion, recursive queries, and so on.

6. **Support for schema evolution**

   The tight coupling between data and applications in an OODBMS makes schema evolution more feasible.

7. **Support for long-duration, transactions**

   Current relational DBMSs enforce serializability on concurrent transactions to maintain database consistency. OODBMSs use a different

protocol to handle the types of long-duration transaction that are common in many advanced database application.

8. **Applicability to advanced database applications**

   There are many areas where traditional DBMSs have not been particularly successful, such as, Computer-Aided Design (CAD), Computer-Aided Software Engineering (CASE), and Multimedia Systems. The enriched modeling capabilities of OODBMSs have made them suitable for these applications.

9. **Improved performance**

   There have been a number of benchmarks that have suggested OODBMSs provide significant performance improvements over relational DBMSs. The results showed an average 30-fold performance improvement for the OODBMS over the RDBMS.

Disadvantages of OODBMSS

The following are disadvantages of OODBMSs:

1. **Lack of universal data model:** There is no universally agreed data model for an OODBMS, and most models lack a theoretical foundation. This disadvantage is seen as a significant drawback, and is comparable to pre-relational systems.
2. **Lack of experience:** In comparison to RDBMSs the use of OODBMS is still relatively limited. This means that we do not yet have the level of experience that we have with traditional systems. OODBMSs are still very much geared towards the programmer, rather than the naïve end-user. Also there is a resistance to the acceptance of the technology. While the OODBMS is limited to a small niche market, this problem will continue to exist

3. **Lack of standards:** There is a general lack of standards of OODBMSs. We have already mentioned that there is not universally agreed data model. Similarly, there is no standard object-oriented query language.
4. **Competition:** Perhaps one of the most significant issues that face OODBMS vendors is the competition posed by the RDBMS and the emerging ORDBMS products. These products have an established user base with significant experience available. SQL is an approved standard and the relational data model has a solid theoretical formation and relational products have many supporting tools to help .both end-users and developers.
5. **Query optimization compromises encapsulations:** Query optimization requires. An understanding of the underlying implementation to access the database efficiently. However, this compromises the concept of incrassation.
6. **Locking at object level may impact performance:** Many OODBMSs use locking as the basis for concurrency control protocol. However, if locking is applied at the object level, locking of an inheritance hierarchy may be problematic, as well as impacting performance.
7. **Complexity:** The increased functionality provided by the OODBMS (such as the illusion of a single level storage model, pointer sizzling, long-duration transactions, version management, and schema evolution--makes the system more complex than that of traditional DBMSs. In complexity leads to products that are more expensive and more difficult to use.
8. **Lack of support for views:** Currently, most OODBMSs do not provide a view mechanism, which, as we have seen previously, provides many advantages such as data independence, security, reduced complexity, and customization.

9. **Lack of support for security:** Currently, OODBMSs do not provide adequate security mechanisms. The user cannot grant access rights on individual objects or classes. If OODBMSs are to expand fully into the business field, these deficiencies must be rectified.

3.3.5 Object Oriented Databases Models

3.3.5.1 Unified Modelling Language

It provides a mechanism in the form of diagrammatic notation and associated language syntax to cover the entire life cycle. Presently, UML can be used by software developers, data modelers, and database designers, and so on to define the detailed specification of an application. They also use it to specify the environment consisting of users, software, communications, and hardware to implement and deploy the application. UML combines commonly accepted concepts from many object-oriented (O-O) methods and methodologies. It is generic, and is language-independent and platform-independent. Software architects can model any type of application, running on any operating system, programming language, or network, in UML.

UML defines nine types of diagrams divided into these two categories:

1. **Structural Diagrams:**
2. These describe the structural or static relationships among schema objects, data objects, and software components. They include class diagrams, object diagrams, component diagrams, and deployment diagrams.
3. **Behavioral Diagrams:** Their purpose is to describe the behavioral or dynamic relationships among components. They include use case diagrams, sequence diagrams, collaboration diagrams, state chart diagrams, and activity diagrams.

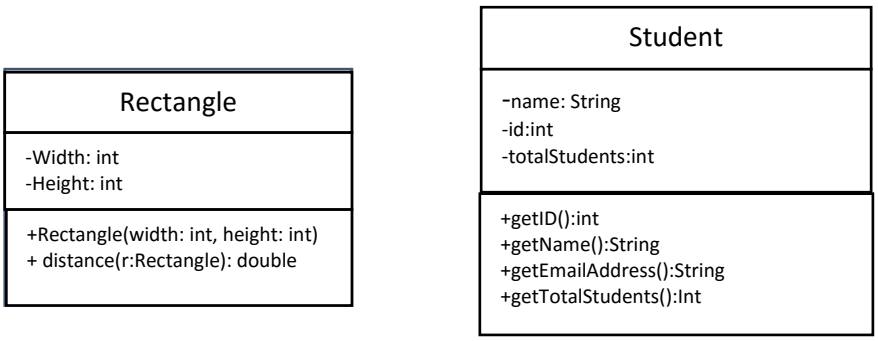
Structural Diagrams

Class Diagrams: Class diagrams capture the static structure of the system and act as foundation for other models. They show classes, interfaces, collaborations, dependencies, generalizations, associations, and other relationships. Class diagrams are a very useful way to model the conceptual database schema.

1. Rules For Forming Class Diagram

1. Class name in the top box
2. Attributes in the middle box. (it usually have the prefix "-")
3. Methods in the bottom box. (it usually have the prefix "+")

4. **Object Diagrams:** Object diagrams show a set of individual objects and their relationships, and are sometimes referred to as *instance diagrams*. They give a static view of a system at a particular time and are normally used to test class diagrams for accuracy.



Component Diagrams: Component diagrams illustrate the organizations and dependencies among software components. A component diagram typically consists of components, interfaces, and dependency relationships. A component may be a source code component, a runtime component, or an executable component. It is a physical building block in the system and is represented as a rectangle with two small rectangles or tabs overlaid on its left side. An **interface** is a group of operations used or created by a component and is usually represented by a small circle. Dependency relationship is used to model the relationship between two components and is represented by a dotted arrow pointing from a component to the component it depends on. For databases, component diagrams stand for stored data such as table spaces or partitions. Interfaces refer to applications that use the stored data.

Deployment Diagrams: Deployment diagrams represent the distribution of components (executables, libraries, tables, files) across the hardware topology. They

depict the physical resources in a system, including nodes, components, and connections, and are basically used to show the configuration of runtime processing elements (the nodes) and the software processes that reside on them (the threads).

Behavioral Diagrams

Use Case Diagrams: Use case diagrams are used to model the functional interactions between users and the system. A **scenario** is a sequence of steps describing an interaction between a user and a system. A **use case** is a set of scenarios that have a common goal. The use case diagram was introduced by Jacobson7 to visualize use cases. A **use case diagram** shows actors interacting with use cases and can be understood easily without the knowledge of any notation. An individual use case is shown as an oval and stands for a specific task performed by the system. An **actor**, shown with a stick person symbol, represents an external user, which may be a human user, a representative group of users, a certain role of a person in the organization, or anything external to the system.

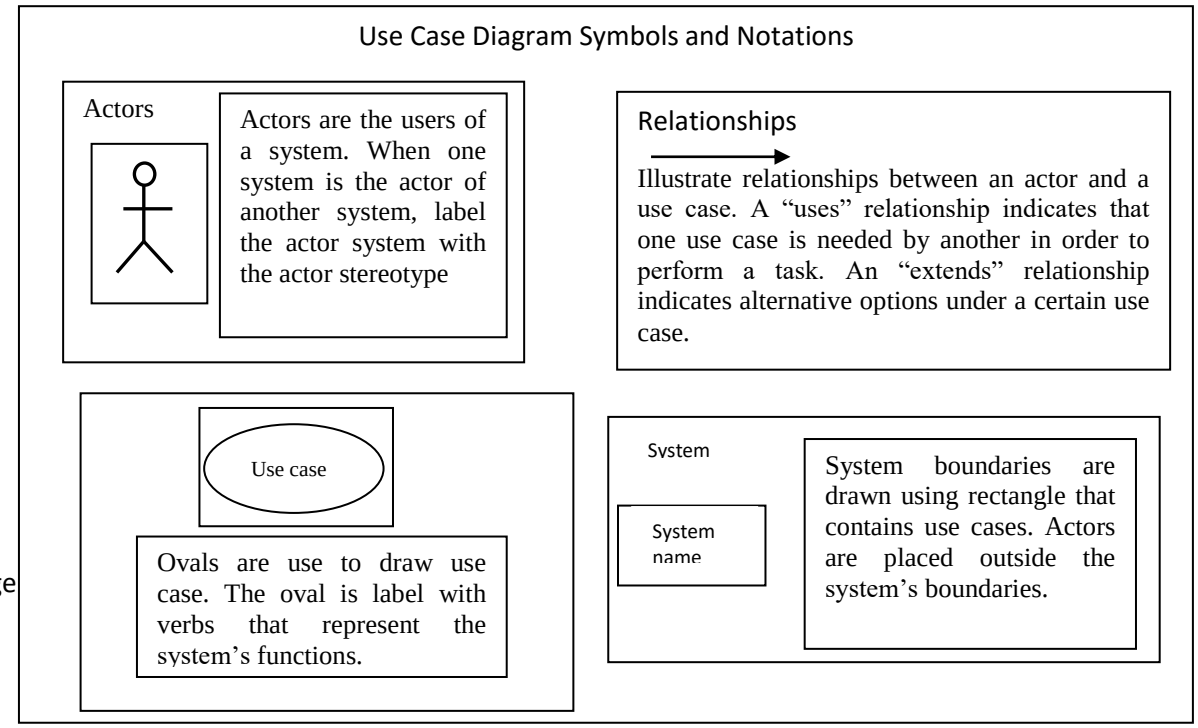


Figure 3.3.1: Symbols and notations of Use Case Diagrams

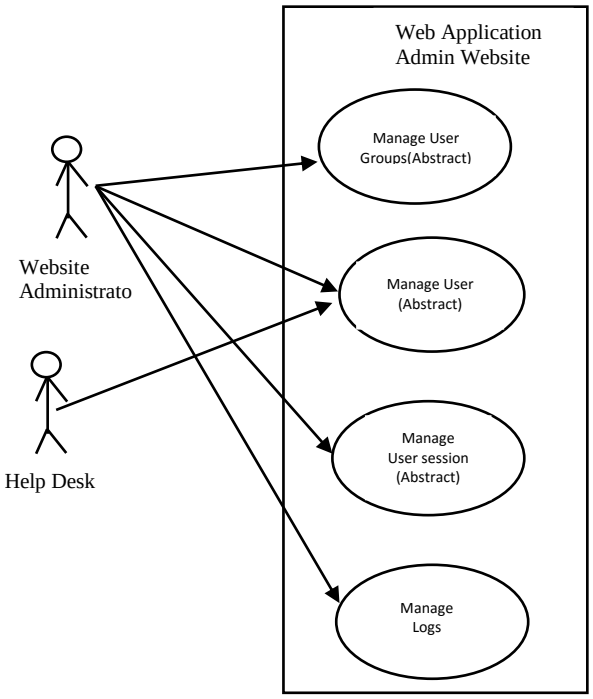


Figure 3.3.2: Use case for a Website Administrator

Sequence Diagrams: Sequence diagrams describe the interactions between various objects over time. They basically give a dynamic view of the system by showing the flow of messages between objects. Within the sequence diagram, an object or an actor is shown as a box at the top of a dashed vertical line, which is called the object's lifeline.

Collaboration Diagrams: Collaboration diagrams represent interactions among objects as a series of sequenced messages. In collaboration diagrams the emphasis is on the structural organization of the objects that send and receive messages, whereas in sequence diagrams the emphasis is on the time-ordering of the messages. Collaboration diagrams show objects as icons and number the messages; numbered messages represent an ordering. The spatial layout of collaboration diagrams allows linkages among objects that show their structural relationships. Use of collaboration and sequence diagrams to represent interactions is a matter of choice as they can be used for somewhat similar purposes; we will hereafter use only sequence diagrams.

Statechart Diagrams: State chart diagrams describe how an object's state changes in response to external events. To describe the behavior of an object, it is common in most object-oriented techniques to draw a state chart diagram to show all the possible states an object can get into in its lifetime. The UML state charts are based on David Harel's state charts. They show a state machine consisting of states, transitions, events, and actions and are very useful in the conceptual design of the application that works against a database of stored objects. State chart diagrams are useful in specifying how an object's reaction to a message depends on its state. An *event* is something done to an object such as receiving a message; an *action* is something that an object does such as sending a message.

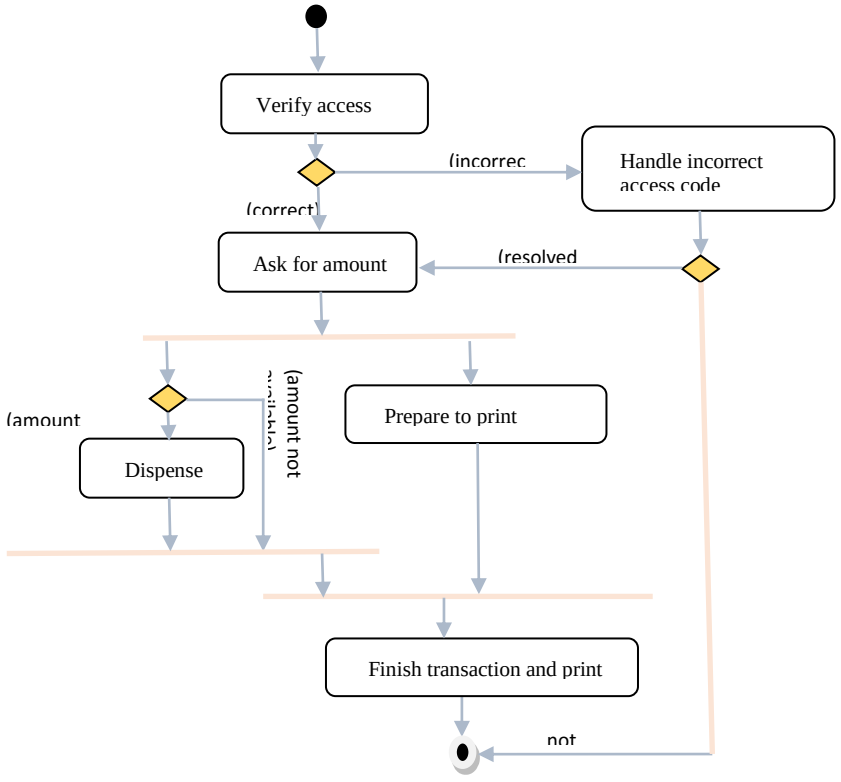


Figure 3.3.3: UML activity diagram: Cash Withdrawal from ATM.

3.3.6 Conclusion

Object-oriented databases are designed and built according to the object-oriented paradigm in which everything is modeled as objects including the data. This type of

data model helps in tackling complex data structures, for instance multimedia content, in a more natural way and provides a seamless transition from design to conception. Hence the data in OODBMS is represented as collection of interacting objects instead of collection of inter-related tables. Usage of object-oriented concepts like polymorphism and inheritance make the interaction between the objects a trivial task. Whereas data is stored as tables in the relational database and we need to relate of "join" tables to perform a query, ***it is stored as a collection of objects in object-oriented database***, and query can be easily performed by following the pointer from parent object to its children.

3.3.7 Tutor marked Assignment

1. How would you define object orientation? What are some of its benefits?
2. What is the difference between an object and a class in the object-oriented data model (OODM)?
3. What are the advantages and disadvantages of OODBMS?

3.3.8 References/ Further Readings

Atkinson M., Bancilhon F. ,Dewitt D. ,Dittrich K., Maier D. and Zdonik S. (1989): The Object Oriented Database Manifesto.

Atkinson, M., et. al., (1989): The object-oriented database system manifesto," in Proc. Int. Conf. On Deductive and Object-Oriented Databases.

Bancilhon, F., (1988): Object Oriented database systems, in Proc. 7th ACM SIGART/SIGMOD Conf.

Bertino, E., Negri, M., Pelagatti, G., and Sbattella, L., (1992): Object-Oriented Query Languages: The Notion and the Issues, IEEE Transactions on Knowledge and Data Engineering, Vol. 4, No. 3.

Hardeep Singh Damesha (2015): Object Oriented Database Management Systems-Concepts, Advantages, Limitations and Comparative Study with Relational Database Management Systems

Kim, W., (1988): A foundation for object-oriented databases", MCC Tech. Rep., N.ACA-ST-248-88.

MODULE 3: DISTRIBUTED DATABASES

UNIT 4: DATABASE AND XML

3.4.1 Introduction

The use of eXtensible Markup Language (XML) is increasing in every field and it is imperative to have a secured place to store the XML documents. XML documents are stored in XML database. XML database is used to store huge amount of information in the XML format. The database can be queried using XQuery. XML is a markup language, which is mainly used to represent structured data. Structured data is the one which contains the data along with the tag / label to indicate what the data is. It is like a data with tag as a column name in RDBMS. Hence the same is used to document the data in DDB. One may think why we need to XML rather than simply documenting the data with simple tags as shown in the contact detail example. XML provides lots of features to handle the structured data within the document

3.4.2 Objectives

At the end of this unit, students should be able to:

4. Understand the various types of XML databases.
5. Understand and be able to state the differences between data-centric and document-centric XML.
6. To understand the principles of Web site management with XML

3.4.3 *What is XML?*

XML is an open and popular standard for marking up text in a way that is both machine and human readable. By “marking up text” we mean that the data in the text files is formatted to include meaningful symbols that communicate to a reader what that data is for. The syntax of XML is similar in style to HTML, the markup language of the World Wide Web (WWW).The data in an XML file can be organized into hierarchies so that the relationships between data elements are visually obvious.

1. XML is the markup language which serves the structured data over the internet, which can be viewed by the user easily as well as quickly.
2. It supports lots of different types of applications.
3. It is easy to write programs which process XMLs.
4. This XML does not have any optional feature so that its complexity can increase. Hence XML is a simple language which any user can use with minimal knowledge.
5. XML documents are created very quickly. It does not need any thorough analysis, design and development phases like in RDBMS. In addition, one should be able to create and view XML in notepad too.
6. All these features of XML make it unique and ideal to represent DDB. A typical XML document begins with <?xml..?>. This is the declaration of xml which is optional, but is important to indicate that it is a xml document. Usually at this beginning line version of the xml is indicated.

1. TYPES OF XML DATABASES

Native XML Database (NXD):

Native XML is a database that:

a. Defines a logical model for an XML document and stores and retrieves documents according to that model.

b. Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.

c. The storage model itself is not constrained. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files.

2. XML Enabled Database (XEDB):

A database that has an added XML mapping layer provided either by the database vendor or a third party. This mapping layer manages the storage and

retrieval of XML data. Data that is mapped into the database is mapped into application specific formats and the original XML meta-data and structure may be lost. Data retrieved as XML is NOT guaranteed to have originated in XML form. Data manipulation may occur via either XML specific technologies (e.g. XPath, XSLT and DOM) or other database technologies (e.g. SQL). The fundamental unit of storage in an XML Enabled Database is implementation dependent.

7. Hybrid XML Databases (HXD):

A database is that can be treated as either a Native XML Database or as an XML Enabled Database depending on the requirements of the application.

Data-Centric Vs. Document-Centric Xml

Deciding whether to use an XML-enabled, native or hybrid XML database is in many cases difficult, and usually depends both on the specific application and the format of the XML documents.

Data-Centric Documents: Data-centric are documents produced as an import or export format, that is, data-centric XML documents are used for machine consumption. These documents are used for communicating data between companies or applications and the fact that XML is used as a common format is simply a matter of convenience, for reasons of interoperability Examples of data-centric documents are sales orders, scientific data, and stock quotes. Since data-centric documents are primarily processed by machines, they have fairly regular structure, fine-grained data and no mixed content.

Document-Centric Documents:

Document-centric are documents usually designed for human consumption, with examples ranging from books to hand-written XHTML documents. They are usually composed directly in XML, or some other format and then converted to XML. Document-centric documents do not need to have regular structure, have coarse-grained data (that is the smallest independent data unit may as well be a document itself) and have mixed content. The examples given above make it clear that the ordering of elements in such documents is always significant.

Storing XML in an XED

XML-enabled databases are primarily used in settings where XML is the format for exchanging data between the underlying database and an application or another database, for example when a Web service is providing data stored in a relational database as an XML document. The main characteristic of the XML-enabled database storage methodology is that it uses a data model other than XML, most commonly the relational data model. Individual instances of this data model are mapped to one or more instances of the XML data model, for example a relational schema can be mapped to different XML schemas depending on the structure of the XML document required by the application that will consume it. As shown in Fig.3.4.1, the XML documents are used as a data exchange format without the XML document to have any particular identity within the database. For example, suppose XML is used to transfer temperature data from a weather station to a database. After the data from a particular document is stored in the database, the document is discarded. To the opposite direction, when an XML document is requested as a result of a query, it is constructed from the results that are retrieved after querying the underlying database, and once it is consumed by the client that has requested it, it is again discarded. There is no way to ask explicitly for a document by its name, nor does any guarantee exist that the original document that was stored in the database can be reconstructed. Because of this, it is not a good idea to shred a document into an XML-enabled database as a way of storing it. The basic use

of XML-enabled systems is for publishing existing relational data (regardless of its source) as XML.

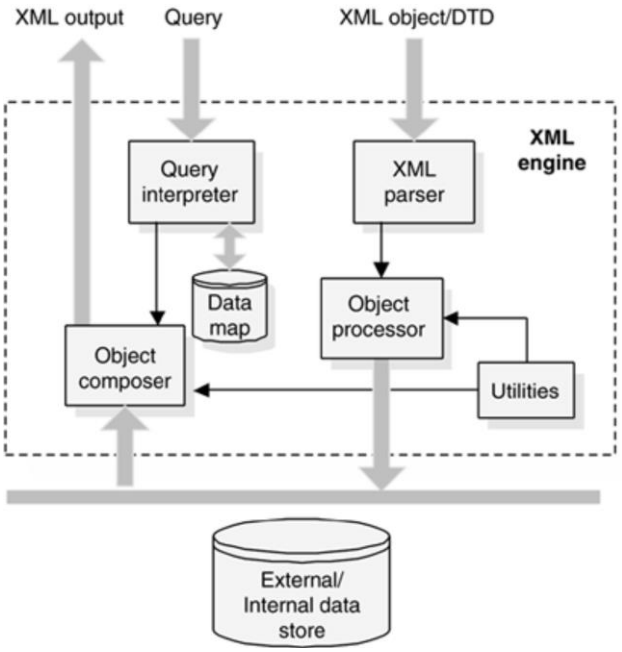


Figure 3.4.1: structure of the XML document

Regardless of whether we are shredding or publishing XML documents, there are two important things to note here. First, an XML-enabled database does not contain XML, i.e. the XML document is completely external to the database. Any XML document/fragment is constructed from data already in the database or is used as a source of new data to store in the database. Second, the database schema matches the XML schema, that is, a different XML schema is needed for each database schema. XML-enabled databases generally include software for performing both publishing and shredding between relational data and XML documents. This extra piece of software can either be integrated into the database or be provided by a third-party vendor outside the database. This software, used by XML-enabled databases, cannot, generally, handle all possible XML documents. Instead, it can handle the subclass of documents that are needed to model the data found in the database.

XML Data Models:

A data model is an abstraction which incorporates only those properties thought to be relevant to the application at hand. As there are different ways in which one can use XML data, there are more than one XML data models. As shown Fig 3.4.2, all XML data models include elements, attributes, text, and document order, but some also include other node types, such as entity references and CDATA sections, while others do not.

Therefore, DTD defines its own XML data model, while XML Schema uses the XML InfoSet data model; XPath and XQuery define their own common data model for querying XML data. This excess of data models makes it difficult for applications to combine different features, such as schema validation together with querying. The W3C is therefore in the process of merging these data models under the XML InfoSet, which is to be replaced by the Post-Schema-Validation Infoset (PSVI).

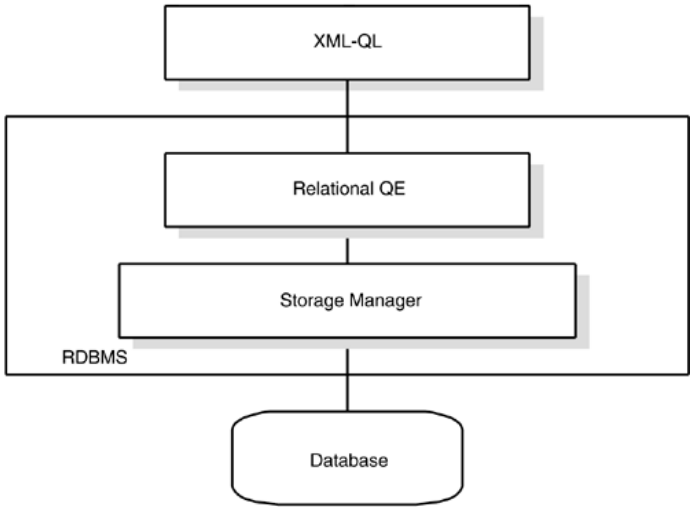


Fig. 3.4.2: Architecture of the XML-Enabled Database

Since many NXDs were created prior to the XML InfoSet and the XPath 2.0 and XQuery data model, they were free to define their own data model. Since the data model of an XML query language defines the minimum amount of information that a native XML database must store, these NXDs need to be upgraded to support XQuery.

Fortunately, the vast majority of NXDs currently supports at least XPath 1.0, and it is envisaged that all of them will support XQuery in the near future.

Relational-to-XML Schema Mapping

When using an XML-enabled database, it is necessary to map the database schema to the XML schema (or vice versa). Such mappings are many-to-many. For example, a database schema for sales order information can be mapped to an XML schema for sales order documents, or it can be mapped to an XML schema for reports showing the total sales by region. There are two important kinds of mappings:

a) Table-based mapping

b) Object-relational mapping

Both table-based mapping and object-relational mappings define bi-directional mappings, that is, the same mapping can be used to transfer data both to and from the database.

Table-Based Mapping:

When using a table-based mapping, the XML document must have the same structure as a relational database. That is, the data is grouped into rows and rows are grouped into \tables". In the following example, the SalesOrders and the Items elements represent the corresponding relational tables containing a list of SalesOrder and Item elements, respectively that in turn represent the rows of each table.

```
<Database>
<SalesOrders>
<SalesOrder>
<Number>123</Number>
<OrderDate>2003-07-28</OrderDate>
<CustomerNumber>456</CustomerNumber>
</SalesOrder>
</SalesOrders>
<Items>
<Item>
<Number>1</Number>
<PartNumber>XY-47</PartNumber>
<Quantity>14</Quantity>
<Price>16.80</Price>
```

```
<OrderNo>123</OrderNo>
</Item> <Item>
<Number>2</Number>
<PartNumber>B-987</PartNumber>
<Quantity>6</Quantity>
 <Price>2.34</Price>
<OrderNo>123</OrderNo>
</Item>
</Items>
 </Database>
```

Object-Relational Mapping:

When using an object-relational mapping, an XML document is viewed as a set of serialized objects and is mapped to the database with an object relational mapping. That is, objects are mapped to tables, properties are mapped to columns and inter-object relationships are mapped to primary key / foreign key relationships. Below is the same document containing sales orders, as above, but encoded using the object-relational mapping. <Database>

```
<SalesOrder>
<Number>123</Number>
<OrderDate>2003-07-28</OrderDate>
<CustomerNumber>456</CustomerNumber>
<Item>
<Number>1</Number>
<PartNumber>XY-47</PartNumber>
<Quantity>14</Quantity>
<Price>16.80</Price>
</Item>
<Item>
<Number>2</Number>
<PartNumber>B-987</PartNumber>
<Quantity>6</Quantity>
<Price>2.34</Price>
</Item>
</SalesOrder>
 </Database>
```

Retrieving and Modifying Query Languages

Query languages in XML-enabled databases are used to extract data from the underlying database and transform it. The most widely used query languages for this purpose, *SQL/XML* and *XQuery*. For XML-enabled relational databases, the most widely used query language is SQL/XML, which provides a set of extensions to SQL for creating XML documents and fragments from relational data and is part of the ISO SQL specification of 2003. The main features of SQL/XML are the provision of an XML data type, a set of scalar functions, XMLELEMENT, XMLATTRIBUTES, XMLFOREST, and XMLCONCAT, and an aggregate function, XMLAGG. For example, the following call to the XMLELEMENT function:

XMLELEMENT (NAME Customer,
XMLELEMENT (NAME Name, customers.name),
XMLELEMENT (NAME ID, customers.id))
constructs the following Customer element for each row in the customers table:
<Customer>
<Name>customer name
</Name>
<ID>customer id</ID>
</Customer>

Updating Xml-Enabled Databases

The solutions related to updating XML data in XML-enabled databases vary from implementation to implementation, due to the idiosyncrasies of the storage model. The most common practice is either to use a custom extension of the SQL/XML prototype to support updates or use a, custom again, product specific API to perform the modification operations over the stored XML data. In either case, though, the update action does not happen in-place, i.e. directly on the tables where the XML data is stored, but rather the document to be updated is extracted from the database, loaded in memory, updated, then returned to the XML-enabling software layer where it is shredded again and stored back to the database.

Fundamental Unit of Storage

A fundamental unit of storage is the smallest grouping of data that logically file together in storage. From a relational database perspective, the fundamental unit of storage is a tuple. From a native XML database perspective, the fundamental unit of storage is a document. Since any document fragment headed by a single element is potentially an XML document, the choice of what constitutes an XML document is purely a design decision.

Text-Based Native Xml Databases

Text-based native XML databases store XML as text. This may be a file within the database itself, a file in the file system outside the database, or a proprietary text format. Note here that the first case implies that relational databases storing XML documents within CLOB (Character Large Object) fields are considered native. All text-based NXDs make use of indices, giving them a big performance advantage when retrieving entire documents or document fragments, as all it takes is a single index-lookup and a single read operation to retrieve the document. This is contrary to XML-enabled databases and some model-based NXDs which require a large number of joins in order to recreate an XML document that has been shredded and inserted in the database. This makes the comparison between text-based NXDs and relational databases analogous to the comparison between hierarchical and relational databases, in that text-based NXDs will outperform relational databases when returning the document or document fragments in the form in which the document is stored; returning data in any other form, such as inverting the document hierarchy, will lead to performance problems.

Model-Based Native Xml Databases

Model-based NXDs have an internal object model and use this to store their XML documents. The way this model is stored varies: one way is to use a relational or object-oriented database; other databases use proprietary storage formats, optimized for their chosen data model. Model-based NXDs built on other databases will have performance similar to those databases, since they rely on these systems to retrieve data; the data

model used, however, can make a notable difference in performance. Model-based NXDs using a proprietary storage format usually have physical pointers between nodes and therefore are expected to have similar performance to text-based NXDs. Clearly, the output format is significant here, as text-based NXDs will probably be faster in outputting in text format, whereas a model-based NXD using the DOM model will be faster in returning a document in DOM format.

Attributes are used to give more meaning to the data represented within the elements. Here attribute indicates what type of contact details are listed like address, phone, email etc.

```
<Contact  category=”ADDRESS”>
<Name> Rose Mathew </Name>
<ApartmentNum>APT 201 </ ApartmentNum>
<AppName> Lakeside terrace 1232 </AppName>
<Street>Lakeside Village Drive </Street>
<Town> Clinton Township </Town>
<State> MI </State>
<Country> US </Country>
</Contact>
```

These elements, attributes are all known as nodes in the document. In short, nodes are the tags / labels in the document. There are seven types of nodes in the XML documents.

1. **Root** : This is the beginning of all the nodes in the document. In our example above contact is the root node.

```
<Contact >
```

2. **Element :** This is the any node in the document that begins with <name> and ends with </name>.

```
<ApartmentNum>APT 201 </ ApartmentNum>
<AppName> Lakeside terrace 1232 </AppName>
```

3. **Text :** This is the value of the element node. In below example, 'Rose Mathew' is a text node.

```
<Name> Rose Mathew </Name>
```

4. **Attribute :** This is the node within the beginning element of the document which specifies more details about the element. It contains name and its value pair always.

```
<Contact  category="ADDRESS">
```

5. **Comment :** This node contains the comment or the description about the data, element or attribute or anything. But it has nothing to do with the actual data. Its only for understanding the document. It is starts with <!-- and ends with --->.

```
<!-- This is the comment node -->
```

6. **Processing Instruction :** This is the node which gives the instruction to the document like sort, display, or anything to do with document. It is always a child node beginning with <? and ending with ?>.

```
<?sort alpha-ascending?>
  <?StudentNames <Fred>, <Bert>, <Harry> ?>
```

7. **Namespace :** Namespace indicates to which bucket the elements belong to. For example, there would same element names used in the document which will have different meaning in their contest – state in address and state for STD code. In order to differentiate this we use **namespace.**

```
<Address: State>
<Phone: State>
```

8. Now it is clear what each state is for. This is similar to appending table name or its alias name before the column names in SQL.

9. These are the very basic things that we need to know while creating an xml document. Let us see how it can be applied in DB.

1. XML elements

10. XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these. Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

Syntax

The following is the syntax to write an XML element –

```
<element-name attribute1 attribute2>
....content
</element-name>
```

where,

1. **element-name** is the name of the element. The *name* its case in the start and end tags must match.
2. **attribute1, attribute2** are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as −

   name = "value"

name is followed by an = sign and a string *value* inside double(" ") or single(' ') quotes.

Empty Element

An empty element (element with no content) has following syntax −

```
<name attribute1 attribute2.../>
```

Following is an example of an XML document using various XML element −

```
<?xml version = "1.0"?>
<contact-info>
   <address category = "residence">
```

```
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 123-4567</phone>
   </address>
</contact-info>
```

XML Elements Rules

Following rules are required to be followed for XML elements −

1. An element *name* can contain any alphanumeric characters. The only punctuation mark allowed in names are the hyphen (-), under-score (_) and period (.).
2. Names are case sensitive. For example, Address, address, and ADDRESS are different names.
3. Start and end tags of an element must be identical.
4. An element, which is a container, can contain text or elements as seen in the above example.

3.4.6 Self-Assessment Question

Is XML a programming language? State reasons to justify your answer.

1. Summary

XML is actually a language used to create other markup languages to describe data in a structured manner. It is a widely supported open technology, (that is a non-proprietary technology) for data exchange and storage.

3.4.7. Conclusion

Processing an XML document requires a software program called an XML Parser or XML**.** Processor. Most XML parsers are available at no charge and for a variety of programming languages (e.g., Java, Perl, C++).

3.4.8 Tutor Marked Assignment:

2. Describe the various types of XML databases
3. Describe the rules for writing XML elements
4. Describe in detail, the purpose of XML

3.4.9 References and Further Reading

Alzarani, H. (2016). Evolution of Object Oriented Database. *Global Journal of Computer Science and Technology: C* .

Ayyasamy, S. B. (May 2017). Performace Evaluation of Native XML database and XML enabled Database. *International Journal of Advanced Research in Computer Science and Software Engineering* .

Balamurugan et al. (2017). Performance Evaluation of Native XML Database and XML Enabled Database. *International Journal of Advanced Research in Computer Science and Software Engg. 7(5),* , 182-191.

Bertino, E. J. (1995). Database security: research and practice. Information systems,.

Gehrke, R. a. (2014). *Database Management Systems, 3ed,* .

George Papamarkos, L. Z. (2010). *XML Database.*

Goebel, V. (2011). *Distributed Database Systems* .

Jorge., D. C. (July 2015). *Basic Principles of Database Security Article* .

Lapis, G. (2005.). *XML and Relational Storage-Are they mutually exclusive* . . IBM Corporation.

Megha, P. T. (2014). An overview of distributed database. *International Journal of Information and Computation Technology.* , 207-214.

Morris, C. C. (2016). *Database System- Implementation and Managemet.*

Morris, C. C. (2017). *DATABASE SYSTEMS.*

N. Derrett, W. K. (1985). Some aspect of oprations in an object oriented database "Database Engineerimg". *IEEE COMPUTER SOCIETY* .

Navathe, R. E. (2010). *FUNDAMENTALS OF Database System: Sixth Edition.* Addison-Wesley.

Özsu, M. T. (June 2014). *Distributed Database Systems.*

T. Atwood. (1985). *"An Object-Oriented DBMS for Design Support Application.*

MODULE 3: DISTRIBUTED DATABASES

UNIT 5: DATA WAREHOUSING

3.5.0 Introduction

Large companies have presences in many places, each of which may generate a large volume of data. For instance, large retail chains have hundreds or thousands of stores, whereas insurance companies may have data from thousands of local branches. Further, large organizations have a complex internal organization structure, and therefore different data may be present in different locations, or on different operational systems, or under different schemas. For instance, manufacturing-problem data and customer-complaint data may be stored on different database systems. Organizations often purchase data from external sources, such as mailing lists that are used for product promotions, or credit scores of customers that are provided by credit bureaus, to decide on credit-worthiness of customers. Corporate decision makers require access to information from multiple such sources. Setting up queries on individual sources is both cumbersome and inefficient. Moreover, the sources of data may store only current data, whereas decision makers may need access to past data as well; for instance, information about how purchase patterns have changed in the past year could be of great importance. Data warehouses provide a solution to these problems.

3.5.1 Objectives

At the end of this unit, students should be able to:

1. To understand data warehousing as an active decision support framework
2. To understand the architecture and components of data ware houses
3. To understand data ware housing schema

3.5.1.1 What is Data Warehouse?

A data warehouse is a repository (or archive) of information gathered from multiple sources, stored under a unified schema, at a single site. A data warehouse is simply a

single, complete, and consistent store of data obtained from a variety of sources and made available to end users in a way they can understand and use it in a business context. Once gathered, the data are stored for a long time, permitting access to historical data. Thus, data warehouses provide the user a single consolidated interface to data, making decision-support queries easier to write. Moreover, by accessing information for decision support from a data warehouse, the decision maker ensures that online transaction-processing systems are not affected by the decision-support workload. *A data warehouse is a subject-oriented, integrated, time variant, and nonvolatile collection of data in support of management's decision-making process.*

Subject Oriented:

1. Organized around major subjects, such as customer, product, sales.
2. Focusing on the modeling and analysis of data for decision makers, not on daily operations or transaction processing.
3. Provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process.

Integrated

1. Data on a given subject is defined and stored once
2. Constructed by integrating multiple, heterogeneous data sources – relational databases, flat files, on-line transaction records
3. One set of consistent, accurate, quality information
4. Standardization
   - Naming conventions
   - Coding structures
   - Data attributes
   - Measures

1. Data cleaning and data integration techniques are applied.
   - Ensure consistency in naming conventions, encoding structures, attribute measures, etc. among different data sources
   - When data is moved to the warehouse, it is converted

Time Variant

Data is stored as a series of snapshots, each representing a period of time

The time horizon for the data warehouse is significantly longer than that of operational systems.

– Operational database: current value data

– Data warehouse data: provide information from a historical perspective (example, past 5-10 years)

• **Every key structure in the data warehouse**

– Contains an element of time, explicitly or implicitly

– But the key of operational data may or may not contain "time element"

Non-Volatile

1. Typically data in the data warehouse is not deleted
2. A physically separate store of data transformed from the operational environment
3. Operational update of data does not occur in the data warehouse environment
4. Does not require transaction processing, recovery, and concurrency control mechanisms
5. Requires only two operations in data accessibility

3.5.2 Components of a Data Warehouse

Figure 3.5.1 shows the architecture of a typical data warehouse, and illustrates the gathering of data, the storage of data, and the querying and data analysis support. Among the issues to be addressed in building a warehouse are the following:

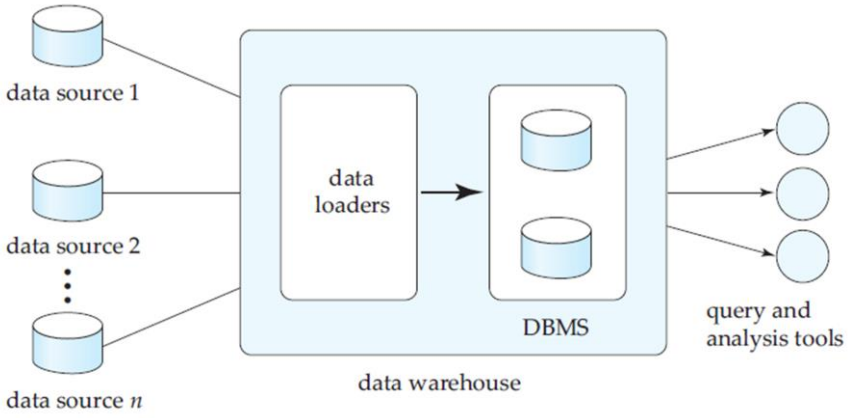


Figure 3.5.1: Architecture of a Data Warehouse

1. **When and how to gather data.** In a **source-driven architecture** for gathering data, the data sources transmit new information, either continually (as transaction processing takes place), or periodically (nightly, for example). In a **destination-driven architecture**, the data warehouse periodically sends requests for new data to the sources. Unless updates at the sources are replicated at the warehouse via two phase commit, the warehouse will never be quite up-to-date with the sources. Two-phase commit is usually far too expensive to be an option, so data warehouses typically have slightly *out-of-date* data. That, however, is usually not a problem for decision-support systems.
2. **What schema to use.** Data sources that have been constructed independently are likely to have different schemas. In fact, they may even use different data models. Part of the task of a warehouse is to perform schema integration, and to convert data to the integrated schema before they are stored. As a result, the data stored in the warehouse are not just a copy of the data at the sources. Instead, they can be thought of as a materialized view of the data at the sources.

3. **Data transformation and cleansing.** The task of correcting and preprocessing data is called **data cleansing**. Data sources often deliver data with numerous minor inconsistencies, which can be corrected. For example, names are often misspelled, and addresses may have street, area, or city names misspelled, or postal codes entered incorrectly. These can be corrected to a reasonable extent by consulting a database of street names and postal codes in each city. The approximate matching of data required for this task is referred to as **fuzzy lookup**.

4. Address lists collected from multiple sources may have duplicates that need to be eliminated in a **merge–purge operation** (this operation is also referred to as

deduplication). Records for multiple individuals in a house may be grouped together so only one mailing is sent to each house; this operation is called ***householding***.

5. Data may be **transformed** in ways other than cleansing, such as changing the units of measure, or converting the data to a different schema by joining data from multiple source relations. Data warehouses typically have graphical tools to support data transformation. Such tools allow transformation to be specified as boxes, and edges can be created between boxes to indicate the flow of data. Conditional boxes can route data to an appropriate next step in transformation.

6. **How to propagate updates**. Updates on relations at the data sources must be propagated to the data warehouse. If the relations at the data warehouse are exactly the same as those at the data source, the propagation is straightforward.

7. **What data to summarize.** The raw data generated by a transaction-processing system may be too large to store online. However, we can answer many queries by maintaining just summary data obtained by aggregation on a relation, rather than maintaining the entire relation. For example, instead of storing data about every sale of clothing, we can store total sales of clothing by item name and category.

8. The different steps involved in getting data into a data warehouse are called **extract, transform, and load** or **ETL** tasks; extraction refers to getting data from the sources, while load refers to loading the data into the data warehouse.

3.5.3 Warehouse Schemas

Data warehouses typically have schemas that are designed for data analysis, using tools such as OLAP tools. Thus, the data are usually multidimensional data, with dimension attributes and measure attributes. Tables containing multidimensional data are called **fact tables** and are usually very large. A table recording sales information for a retail store, with one tuple for each item that is sold, is a typical example of a fact table. The dimensions of the *sales* table would include what the item is (usually an item identifier such as that used in bar codes), the date when the item is sold, which location (store)

the item was sold from, which customer bought the item, and so on. The measure attributes may include the number of items sold and the price of the items.

To minimize storage requirements, dimension attributes are usually short identifiers that are foreign keys into other tables called **dimension tables**. For instance, a fact table *sales* would have attributes *item id*, *store id*, *customer id*, and *date*, and measure attributes *number* and *price*. The attribute *store id* is a foreign key into a dimension table *store*, which has other attributes such as store location (city, state, country). The *item id* attribute of the *sales* table would be a foreign key into a dimension table *item info*, which would contain information such as the name of the item, the category to which the item belongs, and other item details such as color and size. The *customer id* attribute would be a foreign key into a *customer* table containing attributes such as name and address of the customer. We can also view the *date* attribute as a foreign key into a *date info* table giving the month, quarter, and year of each date.

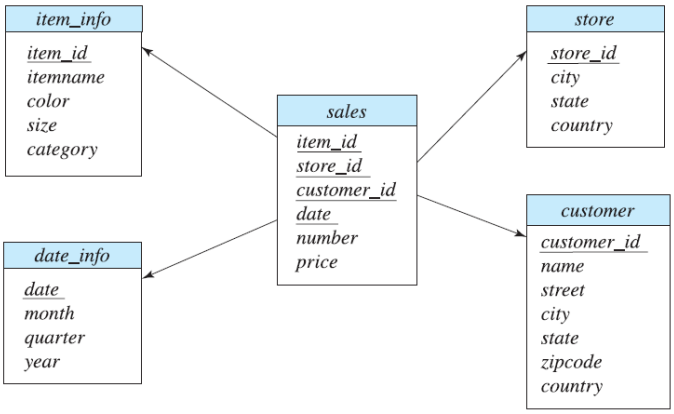


Figure 3.5.2: Star schema fora Data Warehouse

The resultant schema appears in Figure 3.5.2. Such a schema, with a fact table, multiple dimension tables, and foreign keys from the fact table to the dimension tables, is called a **star schema**. More complex data-warehouse designs may have multiple levels of dimension tables; for instance, the *item info* table may have an attribute *manufacturer id*

that is a foreign key into another table giving details of the manufacturer. Such schemas are called **snowflake schemas**. Complex data warehouse designs may also have more than one fact table.

3.5.4 Conclusion:

Decision-support systems analyze online data collected by transaction-processing systems, to help people make business decisions. Since most organizations are extensively computerized today, a very large body of information is available for decision support. Decision-support systems come in various forms, including OLAP systems and data-mining systems. Data warehouses help gather and archive important operational data. Warehouses are used for decision support and analysis on historical data, for instance, to predict trends. Data cleansing from input data sources is often a major task in data warehousing. Warehouse schemas tend to be multidimensional, involving one or a few very large fact tables and several much smaller dimension tables

3.5.5 Tutor Marked Assignment

1. What is data Warehouse?
2. Describe the main characteristics of a star schema
3. With suitable illustration describe the components of a data warehouse

3.5.6 References and Further Readings

Devlin, B., & Cote, L. D. (1996). *Data warehouse: from architecture to implementation*. Addison-Wesley Longman Publishing Co., Inc..

Watson, H. J., & Gray, P. (1997). *Decision support in the data warehouse*. Prentice Hall Professional Technical Reference.

Inmon, W. H. (1996). The data warehouse and data mining. *Communications of the ACM*, *39*(11), 49-51.

Poe, V., Brobst, S., & Klauer, P. (1997). *Building a data warehouse for decision support*. Prentice-Hall, Inc..

Kimball, R., & Merz, R. (2000). The data webhouse toolkit: Building the Web-enabled data warehouse. *Industrial Management & Data Systems*.

MODULE 3: DISTRIBUTED DATABASES

UNIT 6: INTRODUCTION TO DATA MINING

3.6.1 INTRODUCTION

In our day-to-day activities, we generate a lot of data that were previously difficult to store but with the advent of computers, the problem of storage is eliminated. These data are stored on disparate structures and keep increasing by the day. This issue led to the creation of structured databases and database management systems. Managing of these data efficiently and effectively need an effective management system. Database management systems are used in managing large corpus of data in terms of storage and fast retrieval.

Today, data from business transactions, science, medical, personal, surveillance video, satellite sensing, games, digital media, virtual worlds, software engineering, the World Wide Web repositories, text reports and memos are all proliferated. There is need for automatic summarization of data, extraction of the "essence" of information stored and the discovery of patterns in raw data. Unfortunately, these massive collections of data stored on disparate structures very rapidly became overwhelming. This initial chaos has led to the creation of structured databases and DBMS. The efficient database management systems have been very important assets for management of a large corpus of data and especially for effective and efficient retrieval of particular information from a large collection whenever needed. The proliferation of database management systems has also contributed to recent massive gathering of all sorts of information. Today, we have far more information than we can handle: from business transactions and scientific data, to satellite pictures, text reports and military intelligence. Information retrieval is simply not enough anymore for decision-making. Confronted with huge collections of data, we have now created new needs to help us make better managerial choices. These needs are automatic summarization of data, extraction of the essence of information stored, and the discovery of patterns in raw data. These considerations are discussed in this unit.

3.6.2 Objectives*:*

At the end of this unit, students should be able to:

1. understand data mining as a tool for data analysis
2. explore the applications and techniques of data mining
3. understand the concepts knowledge discovery and data mining

3.6.3 What is datamining?

Data mining is the nontrivial extraction of embedded, previously unknown and potentially useful information from data in databases. Data mining (DM) and knowledge discovery in databases (KDD) are interwoven, but data mining is part of knowledge discovery process.

3.6.3.1 Sources of data for KDDD and DM

We have been collecting a myriad of data, from simple numerical measurements and text documents, to more complex information such as spatial data, multimedia channels, and hypertext documents. Here is a non-exclusive list of a variety of information collected in digital form in databases and in flat files.

1. **Business transactions**: Every transaction in the business industry is (often) "memorized" for perpetuity. Such transactions are usually time related and can be inter-business deals such as purchases, exchanges, banking, stock, etc., or intra-business operations such as management of in-house wares and assets. Large department stores, for example, thanks to the widespread use of bar codes, store millions of transactions daily representing often terabytes of data. Storage space is not the major problem, as the price of hard disks is continuously dropping, but the effective use of the data in a reasonable time frame for competitive decision making is definitely the most important problem to solve for businesses that struggle to survive in a highly competitive world.
2. **Medical and personal data**: From government census to personnel and customer files, very large collections of information are continuously gathered about individuals and groups. Governments, companies and organizations such

as hospitals, are stockpiling very important quantities of personal data to help them manage human resources, better understand a market, or simply assist clientele. Regardless of the privacy issues this type of data often reveals, this information is collected, used and even shared. When correlated with other data this information can shed light on customer behaviour and the like.

3. **Surveillance video and pictures**: With the amazing collapse of video camera prices, video cameras are becoming ubiquitous. Video tapes from surveillance cameras are usually recycled and thus the content is lost. However, there is a tendency today to store the tapes and even digitize them for future use and analysis.
4. **Satellite sensing**: There is a countless number of satellites around the globe: some are geo-stationary above a region, and some are orbiting around the Earth, but all are sending a non-stop stream of data to the surface. NASA, which controls a large number of satellites, receives more data every second than what all NASA researchers and engineers can cope with. Many satellite pictures and data are made public as soon as they are received in the hopes that other researchers can analyze them.
5. **Games**: Our society is collecting a tremendous amount of data and statistics about games, players and athletes. From hockey scores, basketball passes and car-racing lapses, to swimming times, boxer's pushes and chess positions, and all the data are stored. Commentators and journalists are using this information for reporting, but trainers and athletes would want to exploit this data to improve performance and better understand opponents.

6. **Digital media**: The proliferation of cheap scanners, desktop video cameras and digital cameras is one of the causes of the explosion in digital media repositories. In addition, many radio stations, television channels and film studios are digitizing their audio and video collections to improve the management of their multimedia assets.

7. **CAD and Software engineering data**: There are a multitude of Computer Assisted Design (CAD) systems for architects to design buildings or engineers to conceive system components or circuits. These systems are generating a tremendous amount of data. Moreover, software engineering is a source of considerable similar data with code, function libraries, objects, etc., which need powerful tools for management and maintenance.
8. **Virtual Worlds**: There are many applications making use of three-dimensional virtual spaces. These spaces and the objects they contain are described with special languages such as VRML. Ideally, these virtual spaces are described in such a way that they can share objects and places. There is a remarkable amount of virtual reality object and space repositories available. Management of these repositories as well as content-based search and retrieval from these repositories are still research issues, while the size of the collections continues to grow.
9. **Text reports and memos (e-mail messages)**: Most of the communications within and between companies or research organizations or even private people, are based on reports and memos in textual forms often exchanged by e-mail. These messages are regularly stored in digital form for future use and reference creating formidable digital libraries.
10. **The WWW repositories**: Since the inception of the WWW in 1993, documents of all sorts of formats, content and description have been collected and inter-connected with hyperlinks making it the largest repository of data ever built. Despite its dynamic and unstructured nature, its heterogeneous characteristic, and it's very often redundancy and inconsistency, the World Wide Web is the most important data collection regularly used for reference because of the broad variety of topics covered and the infinite contributions of resources and publishers. Many believe that the World Wide Web will become the compilation of human knowledge.

3.6.4 KDD Stages

The Knowledge Discovery in Databases process comprises of a steps from raw data collections to some form of new knowledge. It is an iterative process consisting of the following steps:

1. **Data cleaning**: also known as data cleansing, it is a phase in which noise data and irrelevant data are removed from the collection.
2. **Data integration**: at this stage, multiple data sources, often heterogeneous, may be combined in a common source.
3. **Data selection**: at this step, the data relevant to the analysis is decided on and retrieved from the data collection.
4. **Data transformation**: also known as data consolidation, it is a phase in which the selected data is transformed into forms appropriate for the mining procedure.
5. **Data mining**: it is the crucial step in which clever techniques are applied to extract patterns potentially useful.
6. **Pattern evaluation**: in this step, strictly interesting patterns representing knowledge are identified based on given measures.
7. **Knowledge representation**: is the final phase in which the discovered knowledge is visually represented to the user. This essential step uses visualization techniques to help users understand and interpret the data mining results.

It is common to combine some of these steps together. For instance, data cleaning and data integration can be performed together as a pre-processing phase to generate a data warehouse. Data selection and data transformation can also be combined where the consolidation of the data is the result of the selection, or, as for the case of data warehouses, the selection is done on transformed data.

3.6.5 Mining Systems

There are many data mining systems available or being developed. Some are specialized systems dedicated to a given data source or are confined to limited data mining functionalities, other are more versatile and comprehensive. Data mining

systems can be categorized according to various criteria among other classification are the following:

1. **Classification according to the type of data source mined**: this classification categorizes data mining systems according to the type of data handled such as spatial data, multimedia data, time-series data, text data, World Wide Web, etc.
2. **Classification according to the data model drawn on**: this classification categorizes data mining systems based on the data model involved such as *relational database, object-oriented database, data warehouse, transactional*, etc.
3. **Classification according to the kind of knowledge discovered**: this classification categorizes data mining systems based on the kind of knowledge discovered or data mining functionalities, such as characterization, discrimination, association, classification, clustering, etc. Some systems tend to be comprehensive systems offering several data mining functionalities together.
4. **Classification according to mining techniques used**: Data mining systems employ and provide different techniques. This classification categorizes data mining systems according to the data analysis approach used such as machine learning, neural networks, genetic algorithms, statistics, visualization, database oriented or data warehouse-oriented, etc. The classification can also take into account the degree of user interaction involved in the data mining process such as query-driven systems, interactive exploratory systems, or autonomous systems.

3.6.6 Types of mined data

In principle, data mining is not specific to one type of media or data. Data mining should be applicable to any kind of information repository. However, algorithms and approaches may differ when applied to different types of data. Indeed, the challenges presented by different types of data vary significantly. Data mining is being put into use and studied for databases, including relational databases, object-relational databases and object oriented databases, data warehouses, transactional databases, unstructured and semi-structured repositories such as the WWW, advanced databases such as spatial

databases, multimedia databases, time-series databases and textual databases, and even flat files. Here are some examples in more detail:

1. **Flat files**: Flat files are actually the most common data source for data mining algorithms, especially at the research level. Flat files are simple data files in text or binary format with a structure known by the data mining algorithm to be applied. The data in these files can be transactions, time-series data, scientific measurements, etc.
2. **Relational Databases**: Briefly, a relational database consists of a set of tables containing either values of entity attributes, or values of attributes from entity relationships. Tables have columns and rows, where columns represent attributes and rows represent tuples. A tuple in a relational table corresponds to either an object or a relationship between objects and is identified by a set of attribute values representing a unique key.
3. **Data Warehouses**: A data warehouse as a storehouse, is a repository of data collected from multiple data sources (often heterogeneous) and is intended to be used as a whole under the same unified schema. A data warehouse gives the option to analyze data from different sources under the same roof.
4. **Transaction Databases**: A transaction database is a set of records representing transactions, each with a time stamp, an identifier and a set of items. Associated ith the transaction files could also be descriptive data for the items.
5. **Multimedia Databases**: Multimedia databases include video, images, audio and text media. They can be stored on extended object-relational or object-oriented databases, or simply on a file system. Multimedia is characterized by its high dimensionality, which makes data mining even more challenging. Data mining from multimedia repositories may require computer vision, computer graphics, image interpretation, and natural language processing methodologies.
6. **Spatial Databases**: Spatial databases are databases that, in addition to usual data, store geographical information like maps, and global or regional positioning. Such spatial databases present new challenges to data mining algorithms.

7. **Time-Series Databases**: Time-series databases contain time related data such stock market data or logged activities. These databases usually have a continuous flow of new data coming in, which sometimes causes the need for a challenging real time analysis. Data mining in such databases commonly includes the study of trends and correlations between evolutions of different variables, as well as the prediction of trends and movements of the variables in time.
8. **World Wide Web**: The WWW is the most heterogeneous and dynamic repository available. A very large number of authors and publishers are continuously contributing to its growth and metamorphosis, and a massive number of users are accessing its resources daily. Data in the WWW is organized in inter-connected documents. These documents can be text, audio, video, raw data, and even applications. Conceptually, the WWW is comprised of three major components: The content of the Web, which encompasses documents available; the structure of the Web, which covers the hyperlinks and the relationships between documents; and the usage of the web, describing how and when the resources are accessed. A fourth dimension can be added relating the dynamic nature or evolution of the documents. Data mining in the WWW, or web mining, tries to address all these issues and is often divided into web content mining, web structure mining and web usage mining.

3.6.7. DM Tasks and Techniques

The kinds of patterns that can be discovered depend upon the data mining tasks employed. By and large, there are two types of data mining tasks: *descriptive data mining* tasks that describe the general properties of the existing data, and *predictive data mining* tasks that attempt to do predictions based on inference on available data. The data mining functionalities and the variety of knowledge they discover are briefly presented in the following list:

1. **Characterization**: Data characterization is a summarization of general features of objects in a target class, and produces what is called *characteristic rules*. The data relevant to a user-specified class are normally retrieved by a database query and

run through a summarization module to extract the essence of the data at different levels of abstractions.

2. **Discrimination**: Data discrimination produces what are called *discriminant rules* and is basically the comparison of the general features of objects between two classes referred to as the *target class* and the *contrasting class*. The techniques used for data discrimination are very similar to the techniques used for data characterization with the exception that data discrimination results include comparative measures.
3. **Association analysis**: Association analysis is the discovery of what are commonly called *association rules*. It studies the frequency of items occurring together in transactional databases, and based on a threshold called *support*, identifies the frequent item sets. Another threshold, *confidence*, which is the conditional probability than an item appears in a transaction when another item appears, is used to pinpoint association rules. Association analysis is commonly used for market basket analysis.
4. **Classification**: Classification analysis is the organization of data in given classes. Also known as *supervised classification*, the classification uses given class labels to order the objects in the data collection. Classification approaches normally use a *training set* where all objects are already associated with known class labels. The classification algorithm learns from the training set and builds a model. The model is used to classify new objects.
5. **Prediction**: Prediction has attracted considerable attention given the potential implications of successful forecasting in a business context. There are two major types of predictions: one can either try to predict some unavailable data values or pending trends, or predict a class label for some data. The latter is tied to classification. Once a classification model is built based on a training set, the class label of an object can be foreseen based on the attribute values of the object and the attribute values of the classes.
6. **Clustering**: Similar to classification, clustering is the organization of data in classes. However, unlike classification, in clustering, class labels are unknown and

it is up to the clustering algorithm to discover acceptable classes. Clustering is also called *unsupervised classification*, because the classification is not dictated by given class labels. There are many clustering approaches all based on the principle of maximizing the similarity between objects in a same class (*intra-class similarity*) and minimizing the similarity between objects of different classes (*inter-class similarity*).

7. **Outlier analysis**: Outliers are data elements that cannot be grouped in a given class or cluster. Also known as *exceptions* or *surprises*, they are often very important to identify. While outliers can be considered noise and discarded in some applications, they can reveal important knowledge in other domains, and thus can be very significant and their analysis valuable.
8. **Evolution and deviation analysis**: Evolution and deviation analysis pertain to the study of time related data that changes in time. Evolution analysis models evolutionary trends in data, which consent to characterizing, comparing, classifying or clustering of time related data. Deviation analysis, on the other hand, considers differences between measured values and expected values, and attempts to find the cause of the deviations from the anticipated values.

3.6.8 Conclusion

Data mining algorithms embody techniques that have sometimes existed for many years, but have only lately been applied as reliable and scalable tools that time and again outperform older classical statistical methods. DM is the source for automatic summarization of data, extraction of the essence of information stored, and the discovery of patterns in raw data and has affected many domains.

3.6.9 Tutor Marked Assignment

1. Differentiate between data mining and Knowledge Discovery
2. Describe the steps of knowledge discovery
3. Enumerate the type of data that can be mined

3.6.10 References and Further Reading

Frawley, W. J., Piatetsky-Shapiro, G. and Matheus, C. J. (1991). Knowledge Discovery in Databases: An Overview. In G. Piatetsky-Shapiro et al. (eds.), Knowledge Discovery in Databases. AAAI/MIT Press.

J. Han and M. Kamber. (2000). Data Mining: Concepts and Techniques. Morgan Kaufmann.

M. S. Chen, J. Han, and P. S. Yu. (1996.) Data mining: An overview from a database perspective. IEEE Trans. Knowledge and Data Engineering, 8:866-883,

Piateski, G., & Frawley, W. (1991). *Knowledge discovery in databases*. MIT press.

Piatetsky-Shapiro, G. (1996). *Advances in knowledge discovery and data mining* (Vol. 21). U. M. Fayyad, P. Smyth, & R. Uthurusamy (Eds.). Menlo Park: AAAI press.

T. Imielinski and H. Mannila. A database perspective on knowledge discovery. Communications of ACM, 39:58-64, 1996. G. Piatetsky-Shapiro, U. M. Fayyad, and P. Smyth. From data mining to knowledge discovery: An overview. In U.M. Fayyad, et al. (eds.), Advances in Knowledge Discovery and Data Mining, 1-35. AAAI/MIT Press, 1996.

Zaïane, O. R. (1999). Principles of knowledge discovery in databases. *Department of Computing Science, University of Alberta*, *20*.

MODULE 4 – EMERGING DATABASE MODELS, TECHNOLOGIES AND APPLICATIONS

UNIT 1: RELATIONAL AND NON-RELATIONAL DATABASES

1. Introduction

For most of the last 40 years, businesses relied on relational database management systems (RDBMSs)—that used Structured Query Language (SQL) as the programming language. ScaleGrid reports that 60.5% of the commonly used databases are SQL-based RDBMS. Applications in domains such as Multimedia, Geographical Information Systems, and digital libraries demand a completely different set of requirements in terms of the underlying database models. The conventional relational database model is no longer appropriate for these types of data. Furthermore the volume of data is typically significantly larger than in classical database systems. In addition, indexing, retrieving and analyzing data types require specialized functionality not

available in conventional database systems. The unit will cover the some discussions and the requirements and description of emerging databases.

1. Objectives:

At the end of this unit, you should be able to:

1. understand the differences between relational and non-relational databases
2. understand the various database models and their limitations
3. describe the various emerging databases

3.2 SQL-based Database Management Systems:

Relational database management systems (RDBMSs) use SQL, a database management language that offers a highly organized and structured approach to information management. Similar to the way a phone book has different categories of information (name, number, address, etc.) for each line of data, relational databases apply strict, categorical parameters that allow database users to easily organize, access, and maintain information within those parameters. The primary reasons why SQL-based RDBMSs continue to dominate are because they are highly stable and reliable, adhere to a standard that integrates seamlessly with popular software stacks and have been used for more than 40 years. Popular examples of SQL database engines include:

1. Oracle Database
2. MySQL
3. Microsoft SQL Server

3.3 Advantages of RDBMS:

1. ACID compliance: If a database system is "ACID compliant," it satisfies a set of priorities that measure the atomicity, consistency, isolation, and durability of database systems. The more ACID-compliant a database is, the more it serves to guarantee the validity of database transactions, reduce anomalies, safeguard data integrity, and create stable database systems. Generally, SQL-based RDBMSs achieve a high level of ACID compliance, but NoSQL databases give up this distinction to gain speed and flexibility when dealing with unstructured data.

2. Ideal for consistent data systems: With an SQL-based RDBMS, information will remain in its original structure. They offer great speed and stability especially if it does not involve massive amounts of data.
3. Better support options: Because RDBMS databases have been around for over 40 years, it's easier to get support, add-on products, and integrate data from other systems.

1. Disadvantages of RDBMS:

1. **Scalability challenges and difficulties with sharing:** RDBMSs have a more difficult time scaling up in response to massive growth compared to NoSQL databases. These databases also present challenges when it comes to sharing. On the other hand, a non-relational database system (NoSQL-based) handle scaling and growth better than relational databases.
2. Less efficient with NoSQL formats: Most RDBMSs are now compatible with NoSQL data formats, but they don't work with them as efficiently as non-relational databases.
3. Another characteristic of conventional databases is that there are hardy international standards available or used for the content of the databases, being the data that is entered by its users. This typically means that local conventions are applied to limit the diversity of data that may be entered in those databases. As local conventions usually differ from other local conventions this has as disadvantage that data that are entered in one database cannot be compared or integrated with data in other databases, even if those database structures are the same and even if the application domain of the databases is the same.

3.5 Non-Relational Database Systems (NoSQL-based)

When tasked with managing large amounts of unstructured data—like text from emails and customer surveys, data collected by a network of mobile apps, or random social media information. The information is disorganized. There is no clearly-defined schema like you would find an RDBMS. You cannot store it in an RDBMS. But you can with a non relational (or NoSQL) database system. Another reason why non-

relational databases are important is that they work with NoSQL formats like JSON, which has become essential for web-based applications that let websites update "live" without needing to refresh the page.

3.5.1 Advantages of Non-relational database systems:

1. **Excellent for handling "big data" analytics:** The main reason why NoSQL databases are becoming more popular is that they remove the bottleneck of needing to categorize and apply strict structures to massive amounts of information. NoSQL databases like HBase, Cassandra, and CouchDB support the speed and efficiency of server operations while offering the capacity to work with large amounts of data.
2. **No limits on types of data you can store:** NoSQL databases give you unlimited freedom to store diverse types of data in the same place. This offers the flexibility to add new and different types of data to your database at any time.
3. **Easier to scale:** NoSQL databases are easier to scale. They are designed to be fragmented across multiple data centers without much difficulty.
4. **No data preparation required:** When there isn't time to design a complex model, and you need to get a database running fast, non-relational databases save a lot of time.

3.5.2 Disadvantages of Non-Relational database systems:

1. **More difficult to find support:** Because the NoSQL community doesn't have 40 years of history and development behind it, it could be more difficult to find experienced users when you in need support.
2. **Lack of tools:** Another disadvantage relating to newness is that—compared to SQL-based RDBMS solutions—there aren't as many tools to assist with performance testing and analysis.
3. **Compatibility and standardization challenges:** Newer NoSQL database systems also lack the high degree of compatibility and standardization offered by SQL-based alternatives. Therefore, you may find that the data in your non-

relational database management system doesn't readily integrate with other products and services.

1. Types of NoSQL Database engines:

1. Key-value stores, such as Redis and Amazon DynamoDB, are extremely simple database management systems that store only key-value pairs and provide basic functionality for retrieving the value associated with a known key. The simplicity of key-value stores makes these database management systems particularly well-suited to embedded databases, where the stored data is not particularly complex and speed is of paramount importance.
2. Wide column stores, such as Cassandra, Scylla, and HBase, are schema-agnostic systems that enable users to store data in column families or tables, a single row of which can be thought of as a record — a multi-dimensional key-value store. These solutions are designed with the goal of scaling well enough to manage petabytes of data across as many as thousands of commodity servers in a massive, distributed system. Although technically schema-free, wide column stores like Scylla and Cassandra use an SQL variant called CQL for data definition and manipulation, making them straightforward to those already familiar with RDBMS.
3. Document stores, including MongoDB, PostgreDB Couchbase, are schema-free systems that store data in the form of JSON documents. Document stores are similar to key-value or wide column stores, but the document name is the key and the contents of the document, whatever they are, are the value. In a document store, individual records do not require a uniform structure, can contain many different value types, and can be nested. This flexibility makes them particularly well-suited to manage semi-structured data across distributed systems.
4. **Graph databases**, such as Neo4J and Datastax Enterprise Graph, represent data as a network of related nodes or objects in order to facilitate data visualizations and graph analytics. A node or object in a graph database contains free-form data

that is connected by relationships and grouped according to labels. Graph-oriented database management systems (DBMS) software is designed with an emphasis on illustrating connections between data points. As a result, graph databases are typically used when analysis of the relationships between heterogeneous data points is the end goal of the system, such as in fraud prevention, advanced enterprise operations, or Facebook's original friend's graph.

5. **Search engines**, such as elastic search, Splunk, and Solr, store data using schema-free JSON documents. They are similar to document stores, but with a greater emphasis on making your unstructured or semi-structured data easily accessible via text-based searches with strings of varying complexity.

1. Differences between SQL and NoSQL Databases

Having described SQL and NoSQL databases along with their advantages and disadvantages, the table below shows the differences between the two database systems.

Table 4.1: Differences between SQL and NoSQL databases

S/N	SQL	NoSQL
1.	Primary called relational databases	Referred to as non-relational databases
2.	Table based databases	Document based, key-value pairs, graph databases, wide-column stores etc.
3.	Predefined schema for structured data	Dynamic schema for unstructured data
4.	SQL databases are vertically scalable by increasing horse-power of the hardware.	NoSQL databases are horizontally scalable by increasing the database servers in the pool of resources to reduce load.
5.	Structured Query Language (SQL) used for defining and manipulating data.	Unstructured query language is used which varies from database to database.
6.	Emphasizes on ACID properties (Atomicity, Consistency, Isolation and Durability)	Follows the Brewers CAP theorem (Consistency, Availability and Partition Tolerance).

3.8 Conclusion:

Relational databases emerged in the 70s to store data according to a schema that allows data to be displayed as tables with rows and columns. Relational database is a collection of tables, each with a schema that represents the fixed attributes and data types that the items in the table will have. Relational Database Management Systems (RDBMS) provide functionality for reading, creating, updating, and deleting data, typically by means of Structured Query Language (SQL) statements. NoSQL (non-relational) databases emerged as a popular alternative to relational databases as web applications became increasingly complex. NoSQL/non-relational databases can take a variety of forms. However, the critical difference between NoSQL and relational databases is that RDBMS schemas rigidly define how all data inserted into the database must be typed and composed, whereas NoSQL databases can be schema-agnostic, allowing unstructured and semi-structured data to be stored and manipulated.

3.9 References and Further reading

Agrawal, S., & Patel, A. (2016). A Study ON GRAPH STORAGE DATABASE OF NOSQL. *International Journal on Soft Computing, Artificial Intelligence and Applications (IJSCAI)*, *5*(1), 33-39.

https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/

https://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/

https://www.xplenty.com/blog/overview-of-modern-database-systems-which-database-is-right-for-your-use-case/

Karthic, S., & Kandasamy, S. A Java Based XML-Driver for Retrieving Data from the XML Document Using SQL-Query.

Naheman, W., & Wei, J. (2013, December). Review of NoSQL databases and performance testing on HBase. In *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)* (pp. 2304-2309). IEEE.

MODULE 4 – EMERGING DATABASE MODELS, TECHNOLOGIES AND APPLICATIONS

Unit 2: Conventional Database Management Systems

1. Introduction:

Oracle has provided high-quality database solutions since the 1970s. The most recent version of Oracle Database was designed to integrate with cloud-based systems, and it allows you to manage massive databases with billions of records. Moreover, Oracle lets you organize data via a "grid framework" and it uses dynamic data masking for an additional layer of security. Traditionally, Oracle has offered RDMBS solutions, but now you can find SQL and NoSQL database solutions too.

1. Objectives:

1. To better understand the features and capabilities that make a conventional databases
2. To understand latest programming language and multiple data structures support being offered by these conventional database management systems.

4.2.2 ORACLE DATABSE MANAGEMENT SYSTEM

Oracle has provided high-quality database solutions since the 1970s. The most recent version of Oracle Database was designed to integrate with cloud-based systems, and it allows you to manage massive databases with billions of records. Moreover, Oracle lets you organize data via a "grid framework" and it uses dynamic data masking for an additional layer of security. Oracle has offered RDMBS SQL and NoSQL solutions.

4.2.3 Programming Language Support by Oracle:

Many programming languages have Object-Relational Mapping (ORM) frameworks that translate between the object-oriented data structures in programs and the relational model of databases. Such frameworks usually represent themselves as libraries to the developer that are used to abstract data interaction from the application logic. While abstracting the data access layer in this way can boost developer

productivity, the task of managing the abstraction is still the developer's responsibility. Oracle supports a large and varied development community (both internally and externally), so the company recognizes the advantages of broad language support. Today, more than 30 programming languages, including the popular languages shown in Figure 4.1, can access the various database technologies that Oracle provides. In addition, Oracle actively participates in industry-wide efforts to refine standards for database interfaces, including JDBC, Python PEP 249, and PHP, among others.

Figure 4.1: Programming languages supported by Oracle
Source: Gerald Venzl, 2017).

Oracle anticipated the popularity of REST (Representational State Transfer), a technology for creating web services, and the emerging trend of data access abstraction via REST. Oracle introduced a database component that allows RESTful access to data stored in a relational database, document store, or key-value format, making Oracle a pioneer in providing standardized REST interfaces for the data access layer. The component is known as Oracle REST Data Services (ORDS).

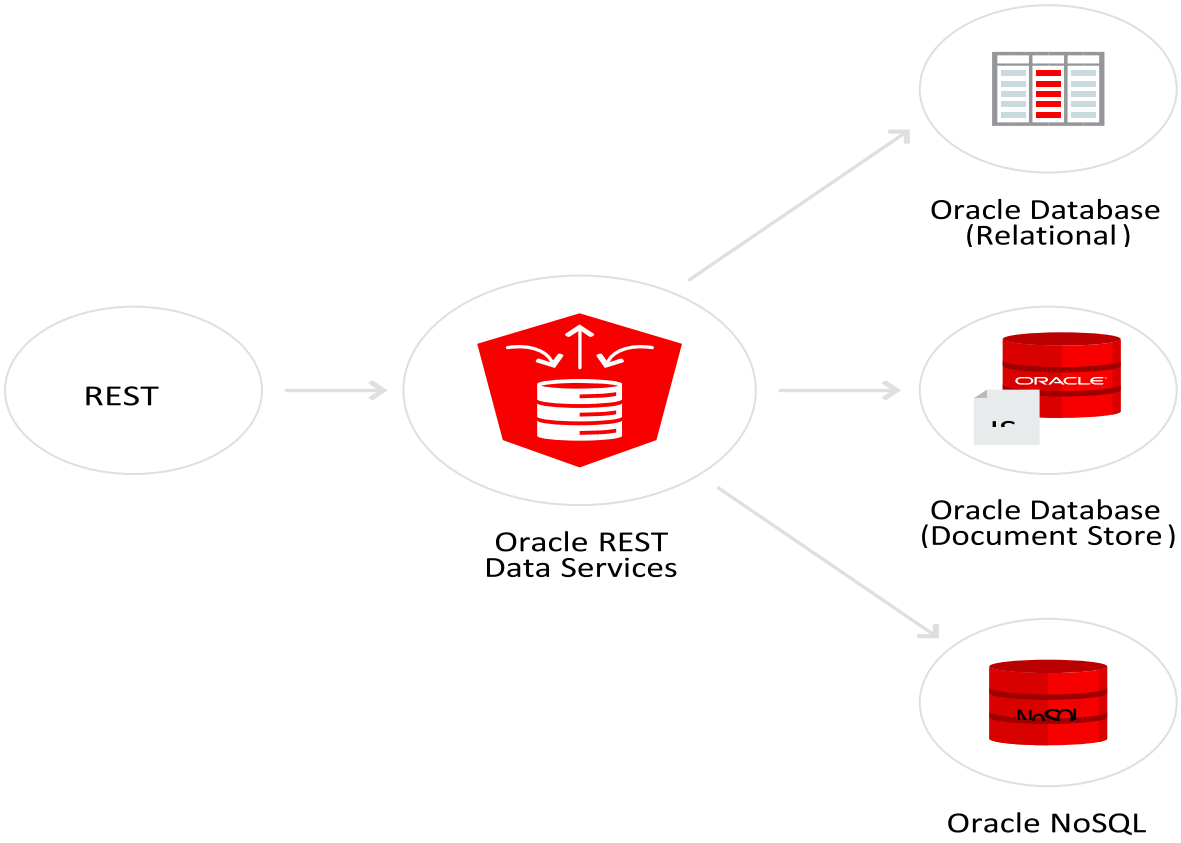


Figure 4.2*: Oracle REST Data Services*

Source: (Gerald Venzl, 2017).

4.2.4 Multi – Model Persistence:

Modern-day databases are often multi-model—that is, they give developers the choice of the desired data model without having to worry about which database to use and how to connect to it (databases that support only a single data model are known as single-model databases). Multi-model databases provide developers with a single connection method and a common API for storing and retrieving data, regardless of the data format. Data storage and retrieval is usually performed via standard SQL operations that provide the desired transparency. By using standard SQL operations, a developer can immediately leverage a multi-model database without having to refactor code or interact with a different set of APIs.

Oracle provides both single-model and multi-model database technologies. Single-model databases from Oracle include (although not exclusively) Berkeley DB, Berkeley

DB XML Data Store, Oracle NoSQL Database, and Oracle *Essbase*. Other databases from Oracle can manage multiple data models, as shown in Figure 2.3. Oracle Database 12c, for example, supports multi-model database capabilities, enabling scalable, high performance data management and analysis.

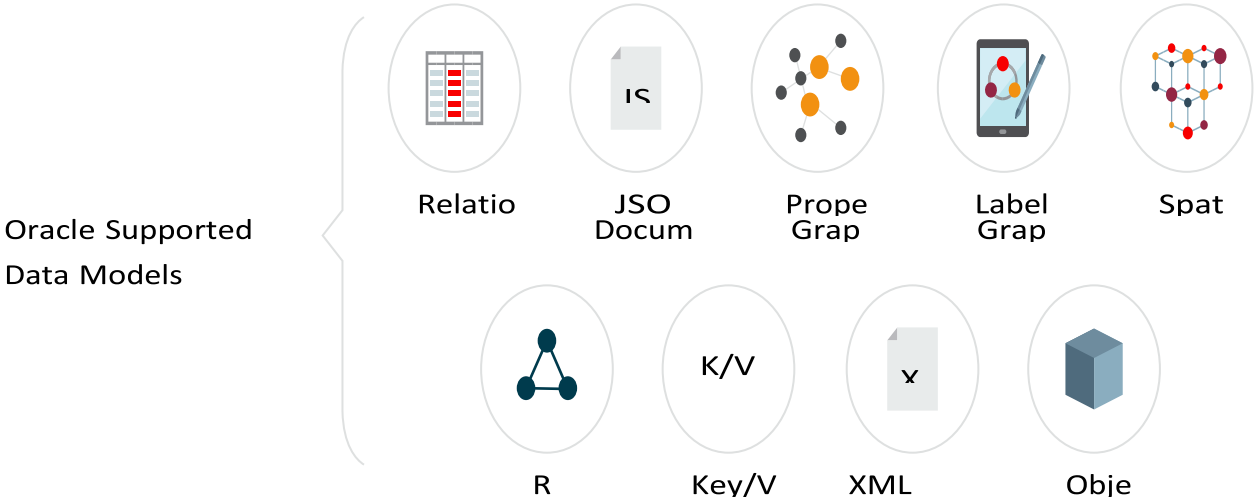


Figure 4.3: Data models supported by Oracle
Source: Gerald Venzl (2017)

4.2.5 GraalVM and the Oracle Database:

GraalVM is a universal virtual machine that runs applications written in a variety of programming languages (JavaScript, Python 3, Ruby, R, Java Virtual Machine (JVM) - based languages, and LLVM-based languages) with high performance on the same platform. Supporting multiple programming languages allows GraalVM to share basic infrastructure such as just-in-time (JIT) compilation, memory management, configuration and tooling among all supported languages. The sharing of configuration and tooling leads to a uniform developer experience in modern projects that make use of multiple programming languages.

Support for many languages is not the only benefit of GraalVM. Although it primarily runs as part of a JVM, GraalVM offers a solution for building native images written in the Java programming language. This native image feature allows to compile an entire Java application, possibly embedding multiple guest languages supported by GraalVM and including the needed virtual machine infrastructure, into a native executable or a

shared library. The main advantages of a GraalVM native image are improved startup times and reduced memory footprint. This enables existing applications and platforms not written in Java to embed GraalVM as a shared library and benefit from its universal virtual machine technology.

GraalVM provides high performance for individual languages and interoperability with zero performance overhead for creating polyglot applications. Instead of converting data structures at language boundaries, GraalVM allows objects and arrays to be used directly by foreign languages.

Example scenarios include accessing functionality of a Java library from Node.js code, calling a Python statistical routine from Java, or using R to create a complex SVG plot from data managed by another language. With GraalVM, programmers are free to use whatever language they think is most productive to solve the current task.

GraalVM 1.0 allows you to run:

1. JVM-based languages like Java, Scala, Groovy, or Kotlin
2. JavaScript (including Node.js)
3. LLVM bitcode (created from programs written in e.g. C, C++, or Rust)
4. Experimental versions of Ruby, R, and Python

GraalVM can either run standalone, embedded as part of platforms like OpenJDK or Node.js, or even embedded inside databases such as MySQL or the Oracle RDBMS. Applications can be deployed flexibly across the stack via the standardized GraalVM execution environments. In the case of data processing engines, GraalVM directly exposes the data stored in custom formats to the running program without any conversion overhead.

For JVM-based languages, GraalVM offers a mechanism to create precompiled native images with instant start up and low memory footprint. The image generation process runs a static analysis to find any code reachable from the main Java method and then performs a full ahead-of-time (AOT) compilation. The resulting native binary contains the whole program in machine code form for immediate execution. It can be

linked with other native programs and can optionally include the GraalVM compiler for complementary JIT compilation support to run any GraalVM-based language with high performance.

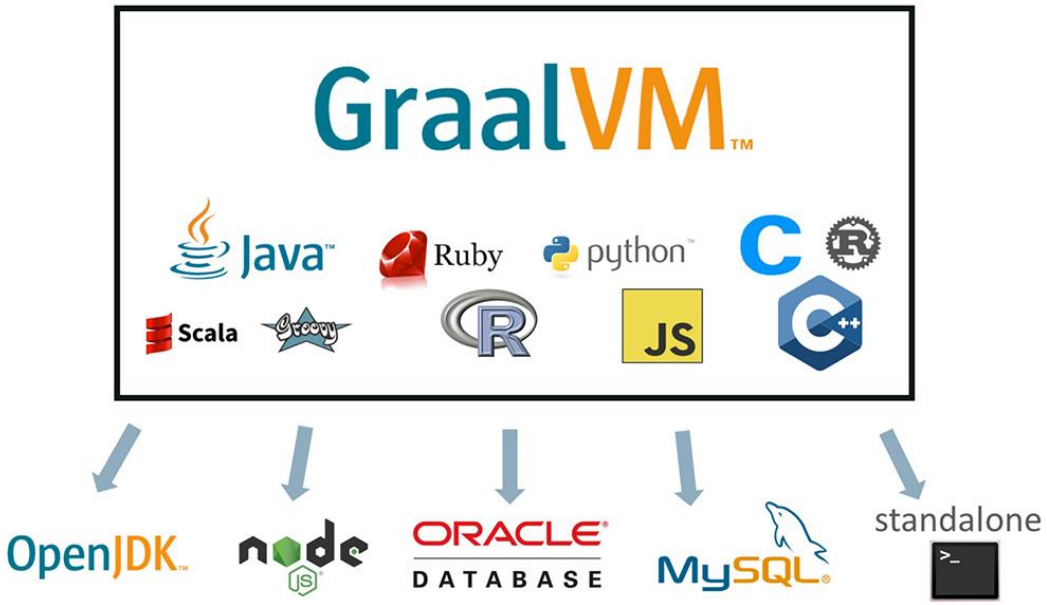


Figure 4.4: GraalVM allows multiple technologies to work with the Oracle Database

Source: https://blogs.oracle.com/developers/announcing-graalvm

A major advantage of the GraalVM ecosystem is language-agnostic tooling that is applicable in all GraalVM deployments. The core GraalVM installation provides a language-agnostic debugger, profiler, and heap viewer. We invite third-party tool developers and language developers to enrich the GraalVM ecosystem using the instrumentation API or the language-implementation API. We envision GraalVM as a language-level virtualization layer that allows leveraging tools and embeddings across all languages.

5.1 MySQL

MySQL is a free, open-source RDBMS solution that Oracle owns and manages. Even though it's freeware, MySQL benefits from frequent security and features updates. Commercial and enterprise can upgrade to paid versions of MySQL to benefit from additional features and user support. Although MySQL didn't support NoSQL in the

past, since Version 8, it provides NoSQL support to compete with other solutions like PostgreSQL.

1. Features of MySQL Database:

The following are the most important properties of MySQL.

1. **Relational Database System:** Like almost all other database systems on the market, MySQL is a relational database system.
2. **Client/Server Architecture:** MySQL is a client/server system. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they query data, save changes, etc. The clients can run on the same computer as the server or on another computer (communication via a local network or the Internet).
3. **SQL compatibility:** MySQL supports as its database language -- as its name suggests – SQL (Structured Query Language). SQL is a standardized language for querying and updating data and for the administration of a database. There are several SQL dialects (about as many as there are database systems). MySQL adheres to the current SQL standard although with significant restrictions and a large number of extensions.
4. **SubSELECTs:** Since version 4.1, MySQL is capable of processing a query in the form SELECT * FROM table1 WHERE x IN (SELECT y FROM table2) (There are also numerous syntax variants for subSELECTs.)
5. **Views:** Put simply, views relate to an SQL query that is viewed as a distinct database object and makes possible a particular view of the database. MySQL has supported views since version 5.0.
6. **Stored procedures** (SPs for short) are generally used to simplify certain steps, such as inserting or deleting a data record. For client programmers this has the advantage that they do not have to process the tables directly, but can rely on SPs. Like views, SPs help in the administration of large database projects. SPs can also increase efficiency. MySQL has supported SPs since version 5.0.

7. **Triggers:** Triggers are SQL commands that are automatically executed by the server in certain database operations (INSERT, UPDATE, and DELETE). MySQL has supported triggers in a limited form from version 5.0, and additional functionality is promised for version 5.1.
8. **Unicode:** MySQL has supported all conceivable character sets since version 4.1, including Latin-1, Latin-2, and Unicode (either in the variant UTF8 or UCS2).
9. **Full-text search:** Full-text search simplifies and accelerates the search for words that are located within a text field. If you employ MySQL for storing text (such as in an Internet discussion group), you can use full-text search to implement simply an efficient search function.
10. **Replication:** Replication allows the contents of a database to be copied (replicated) onto a number of computers. In practice, this is done for two reasons: to increase protection against system failure (so that if one computer goes down, another can be put into service) and to improve the speed of database queries.
11. **Transactions:** In the context of a database system, a transaction means the execution of several database operations as a block. The database system ensures that either all of the operations are correctly executed or none of them. This holds even if in the middle of a transaction there is a power failure, the computer crashes, or some other disaster occurs. Thus, for example, it cannot occur that a sum of money is withdrawn from account A but fails to be deposited in account B due to some type of system error.
12. **Transactions** also give programmers the possibility of interrupting a series of already executed commands (a sort of revocation). In many situations this leads to a considerable simplification of the programming process.
13. **GIS functions:** Since version 4.1, MySQL has supported the storing and processing of two-dimensional geographical data. Thus MySQL is well suited for GIS (geographic information systems) applications.
14. **Programming languages:** There are quite a number of APIs (application programming interfaces) and libraries for the development of MySQL applications.

For client programming you can use, among others, the languages C, C++, Java, Perl, PHP, and Python

15. **ODBC:** MySQL supports the ODBC interface Connector/ODBC. This allows MySQL to be addressed by all the usual programming languages that run under Microsoft Windows (Delphi, Visual Basic, etc.). The ODBC interface can also be implemented under Unix, though that is seldom necessary.
16. Windows programmers who have migrated to Microsoft's new .NET platform can, if they wish, use the ODBC provider or the .NET interface Connector/NET.
17. **Platform independence:** It is not only client applications that run under a variety of operating systems; MySQL itself (that is, the server) can be executed under a number of operating systems. The most important are Apple Macintosh OS X, Linux, Microsoft Windows, and the countless Unix variants, such as AIX, BSDI, FreeBSD, HP-UX, OpenBSD, Net BSD, SGI Iris, and Sun Solaris.
18. **Speed:** MySQL is considered a very fast database program. This speed has been backed up by a large number of benchmark tests (though such tests -- regardless of the source -- should be considered with a good dose of skepticism).

5.3 MySQL User interfaces

A graphical user interface (GUI) is a type of interface that allows users to interact with electronic devices or programs through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation. GUIs are easier to learn than command-line interfaces (CLIs), which require commands to be typed on the keyboard. Third-party proprietary and free graphical administration applications (or "front ends") are available that integrate with MySQL and enable users to work with database structure and data visually. Some well-known front ends are:

1. **MySQL Workbench:** The official integrated environment for MySQL. It was developed by MySQL AB, and enables users to graphically administer MySQL databases and visually design database structures. MySQL Workbench replaces the previous package of software, MySQL GUI Tools. Similar to other third-party

packages, but still considered the authoritative MySQL front end, MySQL Workbench lets users manage database design & modeling, SQL development (replacing MySQL Query Browser) and Database administration (replacing MySQL Administrator).

2. **MySQL Workbench** is available in two editions, the regular free and open source Community Edition which may be downloaded from the MySQL website, and the proprietary Standard Edition which extends and improves the feature set of the Community Edition.

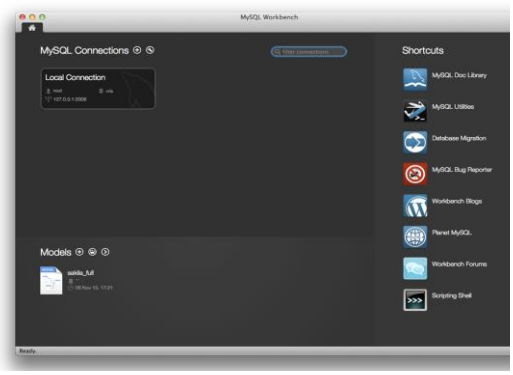


Figure 4.5: MySQL Workbench

Source: Philip Olson - Home computer, MySQL Workbench 5.2.44, GPL, https://commons.wikimedia.org/w/index.php?curid=52195090

Adminer: A free MySQL front end for managing content in MySQL databases (since version 2, it also works on PostgreSQL, Microsoft SQL Server, SQLite and Oracle databases). Adminer is distributed under the Apache License (or GPL v2) in the form of a single PHP file (around 300 KiB in size), and is capable of managing multiple databases, with many CSS skins available. Its author is Jakub Vrána who started to develop this tool as a light-weight alternative to phpMyAdmin.

3. **DBeaver:** An SQL client and a database administration tool. DBeaver includes extended support of following databases: MySQL and MariaDB, PostgreSQL,

Oracle, DB2 (LUW), Exasol, SQL Server, Sybase, Firebird, Teradata, Vertica, Apache Phoenix, Netezza, Informix, Apache Derby, H2, SQLite and any other database which has a JDBC or ODBC driver. DBeaver is free and open source software that is distributed under the Apache License 2.0. The source code is hosted on GitHub.

4. **DBEdit:** A database editor, which can connect to an Oracle, DB2, MySQL and any database that provides a JDBC driver. It runs on Windows, Linux and Solaris. DBEdit is free and open source software and distributed under the GNU General Public License. The source code is hosted on SourceForge.
5. **Navicat:** A series of graphical database management and development software produced by PremiumSoft CyberTech Ltd. for MySQL, MariaDB, Oracle, SQLite, PostgreSQL and Microsoft SQL Server. It has an Explorer-like graphical user interface and supports multiple database connections for local and remote databases. Its design is made to meet the needs of a variety of audiences, from database administrators and programmers to various businesses/companies that serve clients and share information with partners. Navicat is a cross-platform tool and works on Microsoft Windows, OS X and Linux platforms. Upon purchase, users are able to select a language for the software from eight available languages: English, French, German, Spanish, Japanese, Polish, Simplified Chinese and Traditional Chinese.
6. **phpMyAdmin:** A free and open source tool written in PHP intended to handle the administration of MySQL with the use of a web browser. It can perform various tasks such as creating, modifying or deleting databases, tables, fields or rows; executing SQL statements; or managing users and permissions. The software, which is available in 78 languages, is maintained by The phpMyAdmin Project.
7. **Toad for MySQL:** A software application from Dell Software that database developers, database administrators and data analysts use to manage both relational and non-relational databases using SQL. Toad supports many databases and environments. It runs on all 32-bit/64-bit Windows platforms, including

Microsoft Windows Server, Windows XP, Windows Vista, Windows 7 and 8 (32-Bit or 64-Bit). Dell Software has also released a Toad Mac Edition. Dell provides Toad in commercial and trial/freeware versions. The freeware version is available from the ToadWorld.com community.

5.3.1 Command-line interfaces

A command-line interface is a means of interacting with a computer program where the user issues commands to the program by typing in successive lines of text (command lines). MySQL ships with many command line tools, from which the main interface is the mysql client. Some of the command-line utilities available for MySQL are listed below:

1. **MySQL Utilities** is a set of utilities designed to perform common maintenance and administrative tasks. Originally included as part of the MySQL Workbench, the utilities are a stand-alone download available from Oracle.
2. **Percona** Toolkit is a cross-platform toolkit for MySQL, developed in Perl. **Percona** Toolkit can be used to prove replication is working correctly, fix corrupted data, automate repetitive tasks, and speed up servers. Percona Toolkit is included with several Linux distributions such as CentOS and Debian, and packages are available for Fedora and Ubuntu as well. Percona Toolkit was originally developed as Maatkit, but as of late 2011, Maatkit is no longer developed.
3. **MySQL shell** is a tool for interactive use and administration of the MySQL database. It supports JavaScript, Python or SQL modes and it can be used for administration and access purposes.

5.3.2 Advantages of using MySQL:

1. **It's free:** As an open-source RDBMS solution, MySQL is free to use in any way you want.
2. **Excellent for any size organization:** MySQL is an excellent solution for enterprise-level businesses and small startup companies alike.
3. Different user interfaces available

4. Highly compatible with other systems: MySQL has a reputation for being compatible with many other database systems.

5.3.3 Disadvantages of using MySQL:

5. Missing features common in other RDBMSs: Because MySQL prioritizes speed and agility over features, you might find that it's missing some of the standard features found in other solutions, like the ability to create incremental backups, for example.
6. Challenges getting quality support: Being a free solution, there is no dedicated support team to provide solutions for problems and challenges unless it is upgraded to the paid MySQL package from Oracle. However, MySQL does have an active volunteer community, useful forums, and lot of documentation provides answers to questions.

6.0 Microsoft SQL

Microsoft SQL Server is a relational database management system (RDBMS) that supports a wide variety of transaction processing, business intelligence and analytics applications in corporate IT environments. Like other RDBMS software, Microsoft SQL Server is built on top of SQL, a standardized programming language that database administrators (DBAs) and other IT professionals use to manage databases and query the data they contain. SQL Server is tied to Transact-SQL (T-SQL), an implementation of SQL from Microsoft that adds a set of proprietary programming extensions to the standard language. The core component of Microsoft SQL Server is the SQL Server Database Engine, which controls data storage, processing and security. It includes a relational engine that processes commands and queries and a storage engine that manages database files, tables, pages, indexes, data buffers and transactions. Stored procedures, triggers, views and other database objects are also created and executed by the Database Engine.

6.1 Microsoft SQL Services

SQL Server also includes an assortment of add-on services. While these are not essential for the operation of the database system, they provide value added services on top of the core database management system. These services either run as a part of some SQL Server component or out-of-process as Windows Service and presents their own API to control and interact with them.

6.1.1 Machine Learning Services:

The SQL Server Machine Learning services operates within the SQL server instance, allowing people to do machine learning and data analytics without having to send data across the network or be limited by the memory of their own computers. The services come with Microsoft's R and Python distributions that contain commonly used packages for data science, along with some proprietary packages (e.g. revoscalepy, RevoScaleR, microsoftml) that can be used to create machine models at scale. Analysts can either configure their client machine to connect to a remote SQL server and push the script executions to it, or they can run a R or Python scripts as an external script inside a T-SQL query. The trained machine learning model can be stored inside a database and used for scoring.

6.1.2 Service Broker: Used inside an instance, programming environment. For cross-instance applications, Service Broker communicates over TCP/IP and allows the different components to be synchronized, via exchange of messages. The Service Broker, which runs as a part of the database engine, provides a reliable messaging and message queuing platform for SQL Server applications. Service broker services consists of the following parts:

1. message types
2. contracts
3. queues
4. service programs

5. routes

The message type defines the data format used for the message. This can be an XML object, plain text or binary data, as well as a null message body for notifications. The contract defines which messages are used in a conversation between services and who can put messages in the queue. The queue acts as storage provider for the messages. They are internally implemented as tables by SQL Server, but don't support insert, update, or delete functionality. The service program receives and processes service broker messages. Usually the service program is implemented as stored procedure or CLR application. Routes are network addresses where the service broker is located on the network. Also, service broker supports security features like network authentication (using NTLM, Kerberos, or authorization certificates), integrity checking, and message encryption.

6.1.3 Reporting Services:

SQL Server Reporting Services is a report generation environment for data gathered from SQL Server databases. It is administered via a web interface. Reporting services features a web services interface to support the development of custom reporting applications. Reports are created as RDL files. Reports can be designed using recent versions of Microsoft Visual Studio (Visual Studio.NET 2003, 2005, and 2008) with Business Intelligence Development Studio, installed or with the included Report Builder. Once created, RDL files can be rendered in a variety of formats, including Excel, PDF, CSV, XML, BMP, EMF, GIF, JPEG, PNG, and TIFF and HTML Web Archive.

6.3.4 Full Text Search Service:

SQL Server Full Text Search service is a specialized indexing and querying service for unstructured text stored in SQL Server databases. The full text search index can be created on any column with character-based text data. It allows for words to be searched for in the text columns. While it can be performed with the SQL LIKE operator, using SQL Server Full Text Search service can be more efficient. It allows for

inexact matching of the source string, indicated by a Rank value which can range from 0 to 1000—a higher rank means a more accurate match. It also allows linguistic matching ("inflectional search"), i.e., linguistic variants of a word (such as a verb in a different tense) will also be a match for a given word (but with a lower rank than an exact match). Proximity searches are also supported, i.e., if the words searched for do not occur in the sequence they are specified in the query but are near each other, they are also considered a match. T-SQL exposes special operators that can be used to access the FTS capabilities.

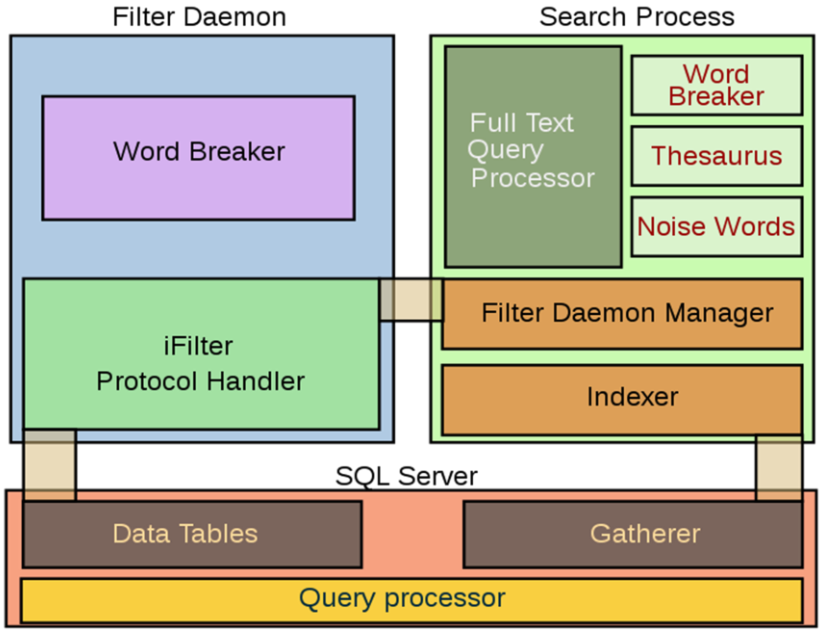


Figure 4.6: SQL Server Full Text Search Architecture

The Full Text Search engine is divided into two processes: The Filter Daemon process (msftefd.exe) and the Search process (msftesql.exe). These processes interact with the SQL Server. The Search process includes the indexer (that creates the full text indexes) and the full text query processor. The indexer scans through text columns in the database. It can also index through binary columns, and use iFilters to extract meaningful text from the binary blob (for example, when a Microsoft Word document is stored as an unstructured binary file in a database). The iFilters are hosted by the Filter Daemon process. Once the text is extracted, the Filter Daemon process breaks it up into a sequence of words and hands it over to the indexer. The indexer filters out

noise words, i.e., words like A, And etc., which occur frequently and are not useful for search. With the remaining words, an inverted index is created, associating each word with the columns they were found in. SQL Server itself includes a Gatherer component that monitors changes to tables and invokes the indexer in case of updates. When a full text query is received by the SQL Server query processor, it is handed over to the FTS query processor in the Search process. The FTS query processor breaks up the query into the constituent words, filters out the noise words, and uses an inbuilt thesaurus to find out the linguistic variants for each word. The words are then queried against the inverted index and a rank of their accurateness is computed. The results are returned to the client via the SQL Server process.

6.1.5 Notification Services:

Originally introduced as a post-release add-on for SQL Server 2000, Notification Services was bundled as part of the Microsoft SQL Server platform for the first and only time with SQL Server 2005. SQL Server Notification Services is a mechanism for generating data-driven notifications, which are sent to Notification Services subscribers. A subscriber registers for a specific event or transaction (which is registered on the database server as a trigger); when the event occurs, Notification Services can use one of three methods to send a message to the subscriber informing about the occurrence of the event.

6.4 Important SQL Server Tools:

6.4.1 SQLCMD:

A command line application that comes with Microsoft SQL Server, and exposes the management features of SQL Server. It allows SQL queries to be written and executed from the command prompt. It can also act as a scripting language to create and run a set of SQL statements as a script. Such scripts are stored as a SQL file, and are used either for management of databases or to create the database schema during the deployment of a database. SQLCMD was introduced with SQL Server 2005 and has continued through SQL Server versions 2008, 2008 R2, 2012, 2014, 2016 and 2019. Its

predecessor for earlier versions was OSQL and ISQL, which were functionally equivalent as it pertains to TSQL execution, and many of the command line parameters are identical, although SQLCMD adds extra versatility.

6.4.2 **Visual Studio:**

A programming tool that includes native support for data programming with Microsoft SQL Server. It can be used to write and debug code to be executed by SQL CLR. It also includes a data designer that can be used to graphically create, view or edit database schemas. Queries can be created either visually or using code. SSMS 2008 onwards, provides intelligence for SQL queries as well.

6.4.3 SQL Server Management Studio:

A Graphic User Interface (GUI) tool included with SQL Server 2005 and later for configuring, managing, and administering all components within Microsoft SQL Server. The tool includes both script editors and graphical tools that work with objects and features of the server. SQL Server Management Studio replaces Enterprise Manager as the primary management interface for Microsoft SQL Server since SQL Server 2005. A version of SQL Server Management Studio is also available for SQL Server Express Edition, for which it is known as SQL Server Management Studio Express (SSMSE).

A central feature of SQL Server Management Studio is the Object Explorer, which allows the user to browse, select, and act upon any of the objects within the server. It can be used to visually observe and analyze query plans and optimize the database performance, among others. SQL Server Management Studio can also be used to create a new database, alter any existing database schema by adding or modifying tables and indexes, or analyze performance. It includes the query windows which provide a GUI based interface to write and execute queries.

6.4.4 SQL Server Operations Studio:

A cross platform query editor available as an optional download. The tool allows users to write queries; export query results; commit SQL scripts to GIT repositories and perform basic server diagnostics. SQL Server Operations Studio supports Windows, Mac and Linux systems. It was released to General Availability in September 2018, at which point it was also renamed to Azure Data Studio. The functionality remains the same as before.

6.4.5 Business Intelligence Development Studio (BIDS): The Integrated Development Environment (IDE) from Microsoft used for developing data analysis and Business Intelligence solutions utilizing the Microsoft SQL Server Analysis Services, Reporting Services and Integration Services. It is based on the Microsoft Visual Studio development environment but is customized with the SQL Server services-specific extensions and project types, including tools, controls and projects for reports (using Reporting Services), Cubes and data mining structures (using Analysis Services. For SQL Server 2012 and later, this IDE has been renamed SQL Server Data Tools (SSDT).

6.5 Advantages of using Microsoft SQL Server include:

1. **Mobilility:** This database engine allows you to access dashboard graphics and visuals via mobile devices.
2. **Integrates with Microsoft products:** Companies that rely heavily on Microsoft products will enjoy the way SQL Server integrates easily with these applications.
3. **Speed:** Microsoft SQL Server has built a reputation around being fast and stable.

6.6 Disadvantages of using Microsoft SQL Server:

1. **Expensive:** Considering that there are plenty of free database engines available, the cost of Microsoft SQL Server is steep. It's over N14,000 for one enterprise-level license per core. There are scaled down licensing options for approximately N3,700 and N900, and a free version you can use to test the platform.

2. **Requires a lot of resources**: This resource-heavy RDBMS may require you to purchase better hardware.

7.1 MongoDB

MongoDB is a free, open-source database engine built especially for applications that use unstructured data. Because most DBMSs were built for structured data—even if add-ons allow them to handle non-relational data now—MongoDB often excels where other DBMSs fail. MongoDB works with structured data too, but since this database engine wasn't designed for relational data, performance slowdowns are likely.

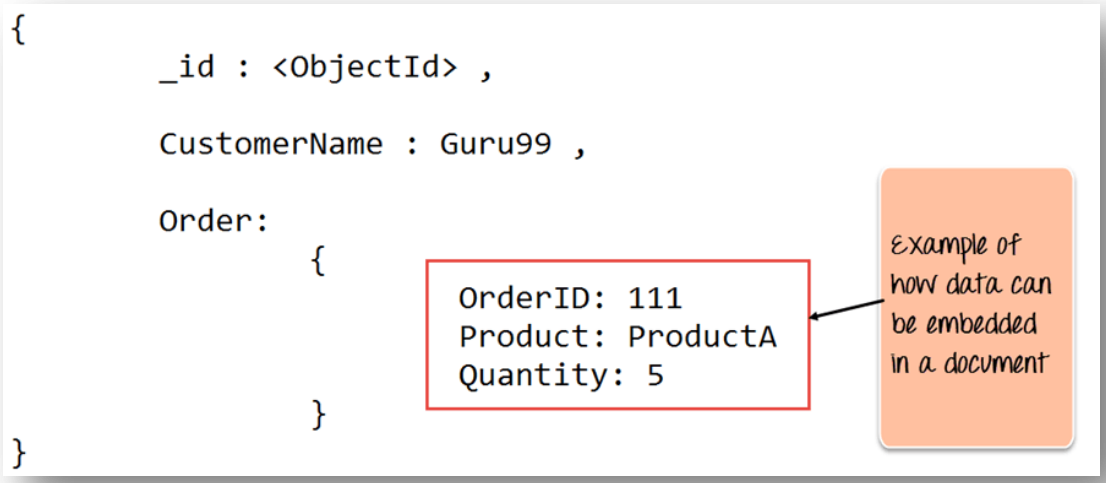
7.2 MongoDB Features:

1. Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
2. The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clear structure with key-value pairs.
3. As seen in the introduction with NoSQL databases, the rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.
4. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
5. **Scalability** – The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database.

7.3 MongoDB Example:

The below example shows how a document can be modeled in MongoDB.

1. The _id field is added by MongoDB to uniquely identify the document in the collection.
2. What you can note is that the Order Data (OrderID, Product, and Quantity) which in RDBMS will normally be stored in a separate table, while in MongoDB it is actually stored as an embedded document in the collection itself. This is one



of the key differences in how data is modelled in MongoDB.

Figure 4.7: MongoDB example
Source: https://www.guru99.com/what-is-mongodb.html

7.4 Key Components of MongoDB Architecture:

The components of MongoDB

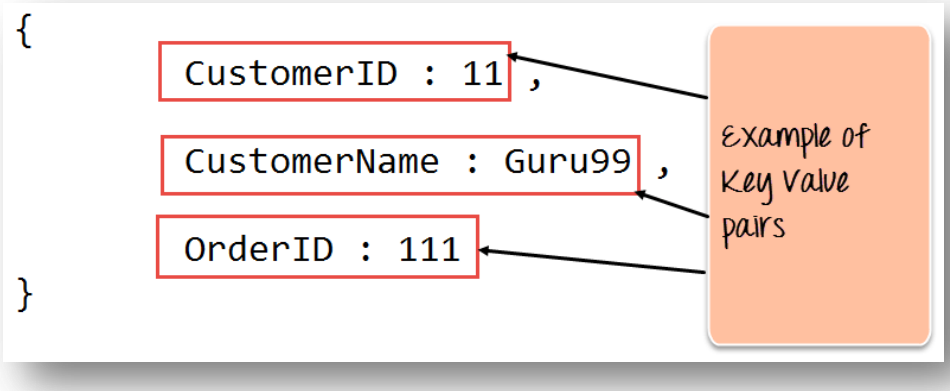
1. **_id** – This is a field required in every MongoDB document. The _id field represents a unique value in the MongoDB document. The _id field is like the document's primary key. If you create a new document without an _id field, MongoDB will automatically create the field. So for example, if we see the example of the above customer table, Mongo DB will add a 24 digit unique identifier to each document in the collection.

Table 5.1: Key features of MongoDB architecture

_Id	CustomerID	CustomerName	OrderID
563479cc8a8a4246bd27d784	11	Guru99	111
563479cc7a8a4246bd47d784	22	Trevor Smith	222
563479cc9a8a4246bd57d784	33	Nicole	333

1. **Collection** – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL. A collection exists within a single database. As seen from the introduction collections don't enforce any sort of structure.
2. **Cursor** – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.
3. **Database** – This is a container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.
4. **Document** - A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.
5. **Field** - A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.

The following diagram shows an example of Fields with Key value pairs. So in the



example below CustomerID and 11 is one of the key value pair's defined in the document.

Figure 4.7: Example of fields with key-value pairs

Source: https://www.guru99.com/what-is-mongodb.html

1. **JSON** – This is known as JavaScript Object Notation. This is a human-readable, plain text format for expressing structured data. JSON is currently supported in many programming languages. Just a quick note on the key difference between the _id field and a normal collection field. The _id field is used to uniquely identify the documents in a collection and is automatically added by MongoDB when the collection is created.

7.5 Why Use MongoDB?:

The reasons as to why one should start using MongoDB

1. **Document-oriented** – Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situation and requirements.

2. **Ad hoc queries** - MongoDB supports searching by field, range queries, and regular expression searches. Queries can be made to return specific fields within documents.
3. **Indexing** - Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.
4. **Replication** - MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.
5. **Load balancing** - MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

7.6 Data Modelling in MongoDB:

As we have seen from the section 7.1, the data in MongoDB has a flexible schema. Unlike in SQL databases, where you must have a table's schema declared before inserting data, MongoDB's collections do not enforce document structure. This sort of flexibility is what makes MongoDB so powerful.

When modeling data in Mongo, keep the following things in mind

1. What are the needs of the application – Look at the business needs of the application and see what data and the type of data needed for the application. Based on this, ensure that the structure of the document is decided accordingly.
2. What are data retrieval patterns – If you foresee a heavy query usage then consider the use of indexes in your data model to improve the efficiency of queries.

3. Are frequent inserts, updates and removals happening in the database – Reconsider the use of indexes or incorporate sharding if required in your data modeling design to improve the efficiency of your overall MongoDB environment.

7.8 Difference between MongoDB & RDBMS

Below are some of the key term differences between MongoDB and RDBMS

Table 5.2: Differences between MongoDB and RDBMS

RDBMS	MongoDB	Difference
Table	Collection	In RDBMS, the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection. The collection contains documents which in turn contains Fields, which in turn are key-value pairs.
Row	Document	In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents.
Column	Field	In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields.
Joins	Embedded documents	In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is sometimes formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. So there is no concept of joins in MongoDB.

Apart from the terms differences, a few other differences are shown below

1. Relational databases are known for enforcing data integrity. This is not an explicit requirement in MongoDB.
2. RDBMS requires that data be normalized first so that it can prevent orphan records and duplicates Normalizing data then has the requirement of more tables, which will then result in more table joins, thus requiring more keys and indexes.

7.9 Conclusion:

MongoDB connects non-relational databases with applications by using a wide variety of drivers (based on the programming language of the application). The most recent versions of MongoDB include pluggable storage engines. Upgraded text search features are also available, along with partial indexing features which can help with performance.

7.10 Tutor Marked Assignment

1. What are the key components of MongoDB?
2. Describe the services provided by Microsoft SQL
3. State the advantages of using Microsoft SQL Server

7.11 Further Reading and References:

Larry Ellison Introduces 'A Big Deal': The Oracle Autonomous Database". Forbes. 2 October 2019. Archived from the original on 22 December 2017. Retrieved 18 December 2019

MSDN: Introducing SQL Server Management Studio". Msdn.microsoft.com. Retrieved 2011-09-04.

Banker, Kyle (March 28, 2011), MongoDB in Action (1st ed.), Manning, p. 375, ISBN 978-1-935182-87-0

Ben-Gan, Itzik, et al. (2006). Inside Microsoft SQL Server 2005: T-SQL Programming. Microsoft Press. ISBN 0-7356-2197-7.

Chodorow, Kristina; Dirolf, Michael (September 23, 2010), MongoDB: The Definitive Guide (1st ed.), O'Reilly Media, p. 216, ISBN 978-1-4493-8156-1

Delaney, Kalen, et al. (2007). Inside SQL Server 2005: Query Tuning and Optimization. Microsoft Press. ISBN 0-7356-2196-9.

Gerald Venzl (2017). Database Requirements for Modern Developers

Giles, Dominic (13 February 2019). "Oracle Database 19c : Now available on Oracle Exadata". Archived from the original on 10 May 2019. Retrieved 10 May 2019.

Hawkins, Tim; Plugge, Eelco; Membrey, Peter (September 26, 2010), The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing (1st ed.), Apress, p. 350, ISBN 978-1-4302-3051-9

Klaus Aschenbrenner (2011). "Introducing Service Broker". Pro SQL Server 2008 Service Broker. Vienna: Apress. p. 17-31. ISBN 978-1-4302-0865-5. Retrieved

Lance Delano, Rajesh George et al. (2005). Wrox's SQL Server 2005 Express Edition Starter Kit (Programmer to Programmer). Microsoft Press. ISBN 0-7645-8923-7.

MariaDB versus MySQL - Compatibility". MariaDB KnowledgeBase. Retrieved 16 September 2016.

MongoDB World". www.mongodb.com. Archived from the original on April 26, 2019. Retrieved April 10, 2019.

MySQL Federated Tables: The Missing Manual". O'Reilly Media. 8 October 2006. Retrieved 1 February 2012.

Pirtle, Mitch (March 3, 2011), MongoDB for Web Development (1st ed.), Addison-Wesley Professional, p. 360, ISBN 978-0-321-7053

Translations". phpMyAdmin. Retrieved 23 December 2019

MODULE 4: EMERGING DATABASE MODELS, TECHNOLOGIES AND APPLICATIONS

UNIT 3: EMERGING DATABASE SYSTEMS

4.3.0 Introduction

The history of databases has types of data stored in the databases were relatively simple. In the past few years, however, there has been an increasing need for handling new data types in databases, such as temporal data, spatial data, multimedia data, and geographic data and so on. Another major trend in the last decade has created its own issues, for example, the growth of mobile computers, starting with laptop computers, palmtop computers and pocket organizers. In more recent years, mobile phones have also come with built-in computers. These trends have resulted into the development of new database technologies to handle new data types and applications. In this unit, some of the emerging database technologies have been briefly introduced.

1. Objectives

At the end of this unit, students should be able to:

1. To understand the basic concept of some emerging databases
2. To understand the design rules and user needs
3. To understand the applications and advantages of each of the databases

4.3.2 EMERGING DATABASE TECHNOLOGIES

This section describes the emerging database technologies:

4.3.2.1 Internet Databases

The Internet revolution of the late 90s have resulted into explosive growth of WWW technology and sharply increased direct user access to databases. Organizations converted many of their phone interfaces to databases into Web interfaces and made a variety of services and information available on-line. The transaction requirements of

organizations have grown with increasing use of computers and the phenomenal growth in the Web technology. These developments have created many sites with millions of viewers and the increasing amount of data collected from these viewers has produced extremely large databases at many companies. Today, millions of people use the Internet to shop for goods and services, listen to music, view network, conduct research, get stock quotes, keep up-to-date with current events and send electronic mail to other Internet users. More and more people are using the Internet at work and at home to view and download multimedia computer files containing graphics, sound, video and text.

Web is used as front end to databases, which can run on any computer system. There is no need to download any special-purpose software to access information. One of the most popular uses of the Web is the viewing, searching and filtering of data. Whether you are using a search engine to find a specific Web site, or browsing Amazon.com's product listings, you are accessing collections of Web-enabled data. Database information can be published on the Web in two different formats: static web publishing and dynamic web publishing.

1. **Static Web publishing** involves creating a list or report, based on the information stored in a database and publishing it online. **Static Web publishing** is a good way to publish information that does not change often and does not need to be filtered or searched.
2. **Dynamic Web publishing** involves creating pages "on the fly" based on information stored in the database. The most commonly used Web database tools for creating Web databases are common gateway interface (CGI) tool and XML.

Advantages of Web Databases

1. Simple to use HTML both for developers and end-users.
2. Platform-independent.
3. Good graphical user interface (GUI).

4. Standardization of HTML.
5. Cross-platform support.
6. Transparent network access.
7. Scalable deployment.
8. Web enables organizations to provide new and innovative services and reach new customers through globally accessible applications

Disadvantages of Web Databases

1. The internet not yet very reliable.
2. Slow communication medium.
3. Security concern.
4. High cost for meeting increasing demands and expectations of customers.
5. Scalability problem due to enormous peak loads.
6. Limited functionality of HTML.

4.3.3 Digital Libraries

The Internet and the WWW are two of the principal building blocks that are used in the development of digital libraries. The Web and its associated technology have been crucial to the rapid growth of digital libraries. This is a fascinating period in the history of libraries and publishing. For the first time, it is possible to build large-scale services where collections of information are stored in digital formats and retrieved over networks. The materials are stored on computers. A network connects the computers to personal computers on the users' desks. In a completely digital library, nothing need ever reach paper. Digital libraries bring together facets of many disciplines and experts with different backgrounds and different approaches.

Digital library can be defined as a managed collection of information, with associated services, where the information is stored in digital formats and accessible over a network. A key part of this definition is that the information is managed. A stream of data sent to

earth from a satellite is not a library. The same data, when organized systematically, becomes a digital library collection. Most people would not consider a database containing financial records of one company to be a digital library, but would accept a collection of such information from many companies as part of a library. Digital libraries contain diverse collections of information for use by many different users. Digital libraries range in size from tiny to huge. They can use any type of computing equipment and any suitable software. The unifying theme is that information is organized on computers and available over a network, with procedures to select the material in the collections, to organize it, to make it available to users and to archive it.

4.3.3.1 Components of Digital Libraries

Digital libraries have the following components:

1. **People**

   Two important communities are the source of much of this innovation. One group is the information professionals. They include librarians, publishers and a wide range of information providers, such as indexing and abstracting services. The other community contains the computer science researchers and their offspring, the Internet developers. Until recently, these two communities had disappointingly little interaction; even now it is commonplace to find a computer scientist who knows nothing of the basic tools of librarianship, or a librarian whose concepts of information retrieval are years out of date. Over the past few years, however, there has been much more collaboration and understanding.

   A variety of words are used to describe the people who are associated with digital libraries. One group of people are the creators of information in the library. Creators include authors, composers, photographers, map makers, designers and anybody else who creates intellectual works. Some are professionals; some are amateurs. Some work individually, others in teams. They have many different reasons for creating

information. Another group is the users of the digital library. Depending on the context, users may be described by different terms. In libraries, they are often called "readers" or "patrons"; at other times they may be called the audience or the customers. A characteristic of digital libraries is that creators and users are sometimes the same people. In academia, scholars and researchers use libraries as resources for their research and publish their findings in forms that become part of digital library collections. The final group of people is a broad one that includes everybody whose role is to support the creators and the users. They can be called information managers. The group includes computer specialists, librarians, publishers, editors and many others. The WWW has created a new profession of Webmaster. Frequently a publisher will represent a creator, or a library will act on behalf of users, but publishers should not be confused with creators or librarians with users. A single individual may be a creator, user and information manager.

2. Economic

Technology influences the economic and social aspects of information and vice versa. The technology of digital libraries is developing fast and so are the financial, organizational and social frameworks. The various groups that are developing digital libraries bring different social conventions and different attitudes to money. Publishers and libraries have a long tradition of managing physical objects, notably books, but also maps, photographs, sound recordings and other artifacts. They evolved economic and legal frameworks that are based on buying and selling these objects. Their natural instinct is to transfer to digital libraries the concepts that have served them well for physical artifacts. Computer scientists and scientific users, such as physicists, have a different tradition. Their interest in digital information began in the days when computers were very expensive. Only a few well-funded researchers had computers on the first networks. They exchanged information informally and openly with

colleagues, without payment. The networks have grown, but the tradition of open information remains.

3. **Computers and networks**

Digital libraries consist of many computers connected by a communications network. The dominant network is the Internet. The emergence of the Internet as a flexible, low-cost, world-wide network has been one of the key factors that have led to the growth of digital libraries.

4.3.3.2 Need for Digital Libraries

The fundamental reason for building digital libraries is a belief that they will provide better delivery of information than was possible in the past. Traditional libraries are a fundamental part of society, but they are not perfect. Computers and networks have already changed the ways in which people communicate with each other. In some disciplines, they argue, a professional or scholar is better served by sitting at a personal computer connected to a communications network than by making a visit to a library. Information that was previously available only to the professional is now directly available to all. From a personal computer, the user is able to consult materials that are stored on computers around the world. Conversely, all but the most diehard enthusiasts recognize that printed documents are so much part of civilization that their dominant role cannot change except gradually. While some important uses of printing may be replaced by electronic information, not everybody considers a large-scale movement to electronic information desirable, even if it is technically, economically and legally feasible.

4.3.3.3 Database for Digital Libraries

4. Digital libraries hold any information that can be encoded as sequences of bits. Sometimes these are digitized versions of conventional media, such as text, images, music, sound recordings, specifications and designs and many, many more. As digital libraries expand, the contents are less often the digital equivalents of

physical items and more often items that have no equivalent, such as data from scientific instruments, computer programs, video games and databases

The information stored in a digital library can be divided into **data and metadata**. Metadata is data about other data. Common categories of metadata include descriptive metadata, such as bibliographic information, structural metadata about formats and structures and administrative metadata, which includes rights, permissions and other information that is used to manage access. One item of metadata is the **identifier**, which identifies an item to the outside world. The distinction between data and metadata often depends upon the context. **Catalogue** records or abstracts are usually considered to be metadata, because they describe other data, but in an online catalogue or a database of abstracts they are the data.

4.3.3.4 Benefits of Digital Libraries

The digital library brings the library to the user:

1. **Using library requires access**. Traditional methods require that the user goes to the library. In a university, the walk to a library takes a few minutes, but not many people are member of universities or have a nearby library. Many engineers or physicians carry out their work with depressingly poor access to the latest information.
2. **A digital library brings the information to the user's desk**, either at work or at home, making it easier to use and hence increasing its usage. With a digital library on the desktop, a user need never visit a library building. The library is wherever there is a personal computer and a network connection.
3. **Computer power is used for searching and browsing:** Computing power can be used to find information. Paper documents are convenient to read, but finding information that is stored on paper can be difficult. Despite the myriad of secondary tools and the skill of reference librarians, using a large library can be a tough

challenge. A claim that used to be made for traditional libraries is that they stimulate serendipity, because readers stumble across unexpected items of value. The truth is that libraries are full of useful materials that readers discover only by accident.

1. **Information can be shared:** Libraries and archives contain much information that is unique. Placing digital information on a network makes it available to everybody. Many digital libraries or electronic publications are maintained at a single central site, perhaps with a few duplicate copies strategically placed around the world. This is a vast improvement over expensive physical duplication of little used material, or the inconvenience of unique material that is inaccessible without traveling to the location where it is stored.

1. **Information is easier to keep current:** Much important information needs to be brought up to date continually. Printed material is awkward to update, since the entire document must be reprinted; all copies of the old version must be tracked down and replaced. Keeping information current is much less of a problem when the definitive version is in digital format and stored on a central computer.

1. **The information is always available:** The doors of the digital library never close; a recent study at a British university found that about half the usage of a library's digital collections was at hours when the library buildings were closed. Material is never checked out to other readers, miss-shelved or stolen; they are never in an off-campus warehouse. The scope of the collections expands beyond the walls of the library. Private papers in an office or the collections of a library on the other side of the world are as easy to use as materials in the local library.

Each of the benefits described above can be seen in existing digital libraries. There is another group of potential benefits, which have not yet been demonstrated, but hold tantalizing prospects. The hope is that digital libraries will develop from static repositories

of immutable objects to provide a wide range of services that allow collaboration and exchange of ideas. The technology of digital libraries is closely related to the technology used in fields such as electronic mail and teleconferencing, which have historically had little relationship to libraries. The potential for convergence between these fields is exciting.

4.3.4 Multimedia databases

Multimedia databases use wide variety of multimedia sources, such as: Images, Video clips, Audio clips, Text or documents. The fundamental characteristics of multimedia systems are that they incorporate continuous media, such as voice (audio), video and animated graphics.

Images include photographs, drawings and so on. Images are usually stored in raw form as a set of pixel or cell values, or in a compressed form to save storage space. The image shape descriptor describes the geometric shape of the raw image, which is typically a rectangle of cells of a certain width and height. Each cell contains a pixel value that describes the cell content. In black/white images, pixels can be one bit. In gray scale or colour images, pixel is multiple bits. Images require very large storages space. Hence, they are often stored in a compressed form, such as GIF, JPEG. These compressed forms use various mathematical transformations to reduce the number of cells stored, without disturbing the main image characteristics. The mathematical transforms used to compress images include Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT) and Wavelet Transforms. In order to identify the particular objects in an image, the image is divided into two homogeneous segments using a homogeneity predicate. The homogeneity predicate defines the conditions for how to automatically group those cells. For example, in a colour image, cells that are adjacent to one another and whose pixel values are close are grouped into a segment. Segmentation and compression can hence identify the main characteristics of an image. Inexpensive image-capture and storage technologies have

allowed massive collections of digital images to be created. However, as a database grows, the difficulty of finding relevant images increases. Two general approach namely manual identification and automatic analysis, to this problem have been developed. Both the approaches use metadata for image retrieval.

2. **Video Clips:** Video clippings include movies, newsreels, and home videos and so on. A video source is typically represented as a sequence of frames, where each frame is a still image. However, rather than identifying the objects and activities in every individual frame, the video is divided into video segments. Each video segment is made up of a sequence of contiguous frames that includes the same objects or activities. Its starting and ending frames identify each segment. The objects and activities identified in each video segment can be used to index the segments. An indexing technique called frame segment trees are used for video indexing. The index includes both objects (such as persons, houses, cars and others) and activities (such as a person delivering a speech, two persons talking and so on). Videos are also often compressed using standards such as MPEG.

3. **Audio Clips:** Audio clips include phone messages, songs, speeches, class presentations, surveillance recording of phone messages and conversations by law enforcement and others. Here, discrete transforms are used to identify the main characteristics of a certain person's voice in order to have similarity based indexing and retrieval Audio characteristic features include loudness, intensity, pitch and clarity.

1. **Text or Documents:** Text or document sources include articles, books, journals and so on. A text or document is basically the full text of some article, book, magazine or journal. These sources are typically indexed by identifying the keywords that appear in the text and their relative frequencies. However, filler words are

eliminated from that process. Because a technique called singular value decompositions (SVD) based on matrix transformation is used to reduce the number of keywords in collection of document. An indexing technique called telescoping vector trees or TV-trees, can then be used to group similar documents together.

4.3.4.1 Multimedia Database Applications

Multimedia data may be stored, delivered and utilized in many different ways. Some of the important applications are as follows:

1. Repository applications.
2. Presentation applications.
3. Collaborative work using multimedia information.
4. Documents and records management.
5. Knowledge dissemination.
6. Education and training.
7. Marketing, advertising, retailing, entertainment and travel.
8. Real-time control and monitoring.

4.3.5 Mobile Databases

The rapid technological development of mobile phones (cell phones), wireless and satellite communications and increased mobility of individual users have resulted into increasing demand for mobile computing. Portable computing devices such as laptop computers, palmtop computers and so on coupled with wireless communications allow clients to access data from virtually anywhere and at any time in the globe. The mobile databases interfaced with these developments, offer the users such as CEOs, marketing professionals, finance managers and others to access any data, anywhere, at any time to take business decisions in real-time. Mobile databases are especially useful to geographically dispersed organizations.

The flourishing of the mobile devices is driving businesses to deliver data to employees and customers wherever they may be. The potential of mobile gear with mobile data is

enormous. A salesperson equipped with a PDA running corporate databases can check order status, sales history and inventory instantly from the client's site. And drivers can use handheld computers to log deliveries and report order changes for a more efficient supply chain.

4.3.5.1. Components of Mobile Databases

Mobile database architecture is a distributed architecture where several computers, generally referred to as corporate database servers are interconnected through a high-speed communication network. Mobile database consists of the following components:

1. Corporate database server and DBMS to manage and store the corporate data and provide corporate applications.
2. Mobile (remote) database and DBMS at several locations to manage and store the mobile data and provide mobile applications.
3. End user mobile database platform consisting of laptop computer, PDA and other Internet access devices to manage and store client (end user) data and provide client applications.

Communication links between the corporate and mobile DBMS for data access. The communication between the corporate and mobile databases is intermittent and is established for short period of time at irregular intervals.

4.3.5.2 Mobile DBMS

The mobile DBMSs are capable of communicating with a range of major relational DBMSs and are providing services that require limited computing resources to match those currently provided by mobile devices. The mobile DBMSs should have the following capabilities:

1. Communicating with centralized or corporate database server through wireless or Internet access.

1. Replicating data on the centralized database server and mobile device.
2. Synchronizing data on the centralized database server and mobile database.
3. Capturing data from various sources such as the Internet.
4. Managing data on the mobile devices such as laptop computer, palmtop computer and so on.
5. Analyzing data on a mobile device.
6. Creating customized mobile applications.

4.3.5.3 Commercial Mobile Databases

Sybase's SQL Anywhere currently dominates the mobile database market. The company has deployed SQL Anywhere, more than 6 million users at over 10,000 sites and serves 68 per cent of the mobile database market, according to a recent Gartner Dataquest study. Other mobile databases include IBM's DB2 Everyplace 7, Microsoft SQL Server 2000 Windows CE Edition and Oracle9i Lite. Smaller player Gupta Technologies' SQLBase also targets handheld devices.

Mobile databases are often stripped-down versions of their server-based counterparts. They contain only basic SQL operations because of limited resources on the devices. In addition to storage requirements for data tables, the database engines require from 125K to 1MB, depending on how well the vendor was able to streamline its code. Platform support is a key issue in choosing a mobile database. No organization wants to devote development and training resources to a platform that may become obsolete. Microsoft's mobile database supports Win32 and Windows CE. The IBM, Oracle and Sybase products support Linux, Palm OS, QNX Neutrino, Symbian EPOC, Windows CE and Win32.

4.3.6 Spatial Databases

Spatial databases keep track of objects in a multidimensional space. Spatial data support in databases is important for efficiently storing, indexing and querying of data based on

spatial locations. A common example of spatial data includes geographic data, such as road maps and associated information. A road map is a two-dimensional object that contains points, lines and polygons that can represent cities, roads and political boundaries such as states or countries. A road map is visualization of graphic information. The location of cities, roads and political boundaries that exist on the surface of the Earth are projected onto two-dimensional display or piece of paper, preserving the relative positions and relative distances of the rendered objects. The data that indicates the Earth location (latitude and longitude, or height and depth) of these rendered objects is the spatial data. When the map is rendered, this spatial data is used to project the locations of the objects on a two-dimensional piece of paper. A geographic information system (GIS) is often used to store, retrieve and render Earth-relative spatial data. Another type of spatial data are the data from computer-aided design (CAD) such as integrated-circuit (IC) designs or building designs and computer-aided manufacturing (CAM) are other types of spatial data. CAD/CAM types of spatial data work on a smaller scale such as for an automobile engine or printed circuit board (PCB) as compared to GIS data, which works at much bigger scale, for example indicating Earth location.

4.3.6.1 Characteristics of Spatial Database

1. A spatial database stores objects that have spatial characteristics that describe them. The spatial relationships among the objects are important and they are often needed when querying the database. A spatial database can refer to an n-dimensional space for any value of ‘n’.
2. Special databases consist of extensions, such as models that can interpret spatial characteristics. In addition, special indexing and storage structures are often needed to improve the performance.
3. The basic extensions needed are to include two-dimensional geometric concepts, such as points, lines and line segments, circles, polygons and arcs, in order to specify the spatial characteristics of the objects.

4. Spatial operations are needed to operate on the objects' spatial characteristics. For example, we need spatial operations to compute the distance between two objects and other such operations. We also need spatial Boolean conditions to check whether two objects spatially overlap and perform other similar operations. For example, a GIS will contain the description of the spatial positions of many types of objects. Some objects such as highways, buildings and other landmarks have static spatial characteristics. Other objects like vehicles, temporary buildings and others have dynamic spatial characteristics that change over time.
5. The spatial databases are designed to make the storage, retrieval and manipulation of spatial data easier and more natural to users such as GIS. Once the data is stored in a spatial database, it can be easily and meaningfully manipulated and retrieved as it relates to all other data stored in the database.
6. Spatial databases provide concepts for databases that keep track of objects in a multi-dimensional space. For example, geographic databases and GIS databases that store maps include two-dimensional spatial descriptions of their objects. These databases are used in many applications, such as environmental, logistics management and war strategies.

4.3.6.2 Spatial Database Queries

Spatial query is the process of selecting features based on their geographic or spatial relationship to other features. There are many types of spatial queries that can be issued to spatial databases. The following categories three typical types of spatial queries.

1. **Range query:** Range query finds the objects of a particular type that are within a given spatial area or within a particular distance from a given location.
2. **Nearest neighbour query or adjacency**: This query finds an object of a particular type that is closest to a given location. For example, finding the police post that is closest to your house, finding all restaurants that lie within five kilometre of

distance of your residence or finding the hospital nearest to the adjacent site and so on.

3. **Spatial joins or overlays:** This query typically joins the objects of two types based on some spatial condition, such as the objects intersecting or overlapping spatially or being within a certain distance

4.3.6.3 Features of spatial databases:

Database systems use indexes to quickly look up values; however, this way of indexing data is not optimal for spatial queries. Instead, spatial databases use a spatial index to speed up database operations. In addition to typical SQL queries such as SELECT statements, spatial databases can perform a wide variety of spatial operations. The following operations and many more are specified by the Open Geospatial Consortium standard:

1. **Spatial Measurements:** Computes line length, polygon area, the distance between geometries, etc.
2. **Spatial Functions:** Modify existing features to create new ones, for example by providing a buffer around them, intersecting features, etc.
3. **Spatial Predicates**: Allows true/false queries about spatial relationships between geometries. Examples include "do two polygons overlap" or 'is there a residence located within a mile of the area we are planning to build the landfill?'
4. **Geometry Constructors:** Creates new geometries, usually by specifying the vertices (points or nodes) which define the shape.
5. **Observer Functions**: Queries which return specific information about a feature such as the location of the center of a circle. Some databases support only simplified or modified sets of these operations, especially in cases of NoSQL systems like MongoDB and CouchDB. In general, spatial data can be of two types:

1. **Vector data:** This data is represented as discrete points, lines and polygons.
2. **Rastor data:** This data is represented as a matrix of square cells

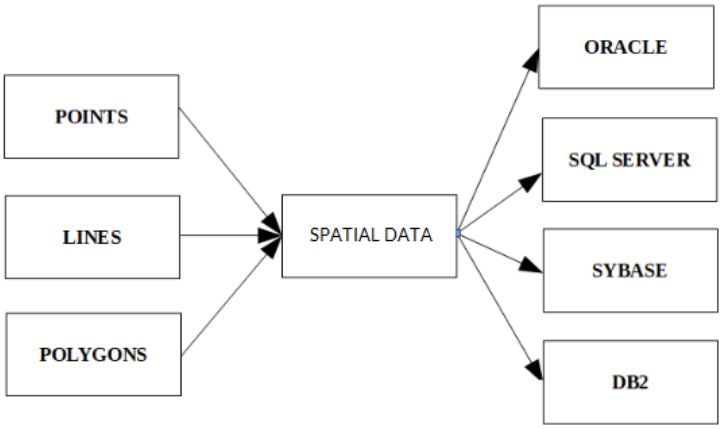


Figure 4.3.1: Spatial Databases

4.3.6.1 Spatial index:

Spatial indices are used by spatial databases (databases which store information related to objects in space) to optimize spatial queries. Conventional index types do not efficiently handle spatial queries such as how far two points differ, or whether points fall within a spatial area of interest. Common spatial index methods include:

1. Geohash
2. HHCode
3. Grid (spatial index)
4. Z-order (curve)
5. Quadtree
6. Octree
7. UB-tree
8. **R-tree:** Typically the preferred method for indexing spatial data. Objects (shapes, lines and points) are grouped using the minimum bounding rectangle (MBR). Objects are added to an MBR within the index that will lead to the smallest increase in its size.

1. R+ tree
2. R* tree
3. Hilbert R-tree
4. X-tree
5. kd-tree

9. **m-tree** – an m-tree index can be used for the efficient resolution of similarity queries on complex objects as compared using an arbitrary metric.
10. Binary space partitioning (BSP-Tree): Subdividing space by hyperplanes

4.3.6.2 Spatial database systems

1. **AllegroGraph** – a graph database which provides a mechanism for efficient storage and retrieval of two-dimensional geospatial coordinates for Resource Description Framework data. It includes an extension syntax for SPARQL queries.
2. **Caliper extends** the Raima Data Manager with spatial datatypes, functions, and utilities.
3. **CouchDB** a document-based database system that can be spatially enabled by a plugin called Geocouch
4. Esri has a number of both single-user and multiuser geodatabases.
5. **GeoMesa** is a cloud-based spatio-temporal database built on top of Apache Accumulo and Apache Hadoop (also supports Apache HBase, Google Bigtable, Apache Cassandra, and Apache Kafka). GeoMesa supports full OGC Simple Features and a GeoServer plugin.
6. **H2 supports** geometry types and spatial indices as of version 1.3.173 (2013-07-28). An extension called H2GIS available on Maven Central gives full OGC Simple Features support.
7. **IBM DB2** Spatial Extender can spatially-enable any edition of DB2, including the free DB2 Express-C, with support for spatial types

8. IBM Informix Geodetic and Spatial datablade extensions auto-install on use and expand Informix's datatypes to include multiple standard coordinate systems and support for RTree indexes. Geodetic and Spatial data can also be incorporated with Informix's Timeseries data support for tracking objects in motion over time.
9. Linter SQL Server supports spatial types and spatial functions according to the OpenGIS specifications.
10. Microsoft SQL Server has support for spatial types since version 2008
11. MonetDB/GIS extension for MonetDB adds OGS Simple Features to the relational column-store database.
12. MySQL DBMS implements the datatype geometry, plus some spatial functions implemented according to the OpenGIS specifications. However, in MySQL version 5.5 and earlier, functions that test spatial relationships are limited to working with minimum bounding rectangles rather than the actual geometries. MySQL versions earlier than 5.0.16 only supported spatial data in MyISAM tables. As of MySQL 5.0.16, InnoDB, NDB, BDB, and ARCHIVE also support spatial features.
13. Neo4j – a graph database that can build 1D and 2D indexes as B-tree, Quadtree and Hilbert curve directly in the graph
14. OpenLink Virtuoso has supported SQL/MM since version 6.01.3126, with significant enhancements including GeoSPARQL in Open Source Edition 7.2.6, and in Enterprise Edition 8.2.0
15. Oracle Spatial
16. PostgreSQL DBMS (database management system) uses the spatial extension PostGIS to implement the standardized datatype geometry and corresponding functions.
17. Redis with the Geo API.
18. RethinkDB supports geospatial indexes in 2D.

19. SAP HANA supports geospatial with SPS08.
20. Smallworld VMDS, the native GE Smallworld GIS database
21. Spatial Query Server from Boeing spatially enables Sybase ASE.
22. SpatiaLite extends Sqlite with spatial datatypes, functions, and utilities.
23. Tarantool supports geospatial queries with RTREE index.
24. Teradata Geospatial includes 2D spatial functionality (OGC-compliant) in its data warehouse system.
25. Vertica Place, the geo-spatial extension for HP Vertica, adds OGC-compliant spatial features to the relational column-store database.

4.3.7 Emerging databases in support of scientific data

4.3.7.1 **Vertical Database**

A vertical database is one in which the physical layout of the data is column-by-column rather than row-by-row. Rather than being arranged in horizontal record structures and processed vertically, data in a vertical database is arranged in vertical structures, known as predicate trees, or P-trees, and processed horizontally. The applications and advantages of vertical databases are as follows:

1. **Data Mining:** Horizontal databases are suitable for applications where the requested result is a set of horizontal records, but less so for applications such as data mining, where researchers are typically interested in results that can be expressed succinctly. P-trees, on the other hand, are well suited to data mining. P-trees are usually created by decomposing each attribute, or column, of a table of horizontal records into separate bit vectors, or array data structures. P-trees can be one-dimensional, two-dimensional or multi-dimensional; if the data to be stored in the database has natural dimensions for instance, geospatial data or geographic information -- the dimensions of the P-tree are matched to those of the data.
2. **Performance:** Data in a vertical database is processed through fast logical operators, such as AND, OR, exclusive OR and complement. Furthermore, by arranging data column-wise rather than row-wise, it is possible to execute queries, or searches, on the data without accessing pages on a hard disk that aren't affected by the query and so increase the speed of data retrieval. This is an important consideration when data mining in very large data repositories.

3. **Page Size:** Another advantage of vertical databases is that they allow data to be stored in large pages. A large page size means that a large number of relevant data items can be retrieved in a single read operation. By contrast, a single read operation on a horizontal database retrieves not only relevant data items, but also attributes, or columns, that aren't relevant to the query in question and favors small page sizes.

4.3.8 MonetDB

MonetDB is an open-source column-oriented database management system developed at the Centrum Wiskunde & Informatica (CWI) in the Netherlands. It was designed to provide high performance on complex queries against large databases, such as combining tables with hundreds of columns and millions of rows. MonetDB has been applied in high-

performance applications for online analytical processing, data mining, geographic information system (GIS), Resource Description Framework (RDF), scientific applications, text retrieval and sequence alignment processing.

4.3.8.1 Components of MonetDB:

A number of extensions exist for MonetDB that extend the functionality of the database engine. Due to the three-layer architecture, top-level query interfaces can benefit from optimizations done in the backend and kernel layers.

1. **SQL:** MonetDB/SQL is a top-level extension, which provides complete support for transactions in compliance with the SQL:2003 standard.
2. **GIS:** MonetDB/GIS is an extension to MonetDB/SQL with support for the Simple Features Access standard of Open Geospatial Consortium (OGC).
3. **SciQL**: SciQL an SQL-based query language for science applications with arrays as first class citizens. SciQL allows MonetDB to effectively function as an array database. SciQL is used in the European Union PlanetData and TELEIOS project, together with the Data Vault technology, providing transparent access to large scientific data repositories. Data Vaults map the data from the distributed repositories to SciQL arrays, allowing for improved handling of spatio-temporal data in MonetDB. SciQL will be further extended for the Human Brain Project.
4. **Data Vaults:** Data Vault is a database-attached external file repository for MonetDB, similar to the SQL/MED standard. The Data Vault technology allows for transparent integration with distributed/remote file repositories. It is designed for scientific data data exploration and mining, specifically for remote sensing data.
5. **R integration:** MonetDB/R module allows for user defined functions (UDFs) written in R to be executed in the SQL layer of the system. This is done using the native R support for running embedded in another application, inside the RDBMS in this case. Previously the MonetDB.R connector allowed the using MonetDB data

sources and process them in an R session. The newer R integration feature of MonetDB does not require data to be transferred between the RDBMS and the R session, reducing overhead and improving performance. The feature is intended to give users access to functions of the R statistical software for in-line analysis of data stored in the RDBMS. It complements the existing support for C UDFs and is intended to be used for in-database processing.

6. **Python integration:** Similarly to the embedded R UDFs in MonetDB, the database now has support for UDFs written in Python/NumPy. The implementation uses Numpy arrays (themselves Python wrappers for C arrays), as a result there is limited overhead - providing a functional Python integration with speed matching native SQL functions. The Embedded Python functions also support mapped operations, allowing user to execute Python functions in parallel within SQL queries.
7. **MonetDBLite:** Following the release of remote driver for R (MonetDB.R) and R UDFs in MonetDB (MonetDB/R), the authors created an embedded version of MonetDB in R called MonetDBLite. It is distributed as an R package, removing the need to manage a database server, required for the previous R integrations. The DBMS runs within the R process itself, eliminating socket communication and serialisation overhead - greatly improving efficiency. The idea behind it is to deliver an SQLite-like package for R, with the performance of an in-memory optimized columnar store.

4.3.9 SciDB

SciDB is a column-oriented database management system (DBMS) designed for multidimensional data management and analytics common to scientific, geospatial, financial, and industrial applications. It is developed by Paradigm4 and co-created by Turing Award winner Michael Stonebraker.

4.3.9.1 Requirements and Features of SciDB

A summary of the requirements of SciDB are as follows:

1. A data model based on multidimensional arrays, not sets of tuples
2. A storage model based on versions and not update in place
3. Built-in support for provenance (lineage), workflows, and uncertainty
4. Scalability to 100s of petabytes and 1,000s of nodes with high degrees of tolerance to failures
5. Support for "external" data objects so that data sets can be queried and manipulated without ever having to be loaded into the database
6. Open source in order to foster a community of contributors and to ensure that data is never locked up — a critical requirement for scientists.

The SciDB team identifies as key features of their eventual product its array-oriented data model, its support for versions, provenance, and time table, its architecture to allow massively parallel computations, scalable on commodity hardware, grids, and clouds, its first-class support for userdefined functions (UDFs), and its native support for uncertainty.

The SciDB data model supports nested multi-dimensional arrays—often a natural representation for spatially or temporally ordered data. Array cells can be tuples, or other arrays, and the type system is extensible. Sparse array representation and operations are supported, with user-definable handling of null or missing data. SciDB allows arrays to be “chunked” (in multiple dimensions) in storage, with chunks partitioned across a collection of nodes. Each node has processing and storage capabilities. Overlaps” are definable so that certain neighborhood operations are possible without communication among nodes. The underlying architectural conception is of a shared-nothing cluster of tens to thousands of nodes on commodity hardware, with a single runtime supervisor

dispatching queries and coordinating execution among the nodes' local executors and storage managers.

An array query language is defined, and refers to arrays as though they were not distributed. A query planner optimizes queries for efficient data access and processing, with a query plan running on a node's local executor/storage manager, and a runtime supervisor coordinating execution. The Array Query Language (AQL) is a declarative SQL-like language with array extensions. There are a number of array-specific operators, and linear algebraic and matrix operations are provided. The language is extensible with Postgres-style user-defined functions, and interfaces to other packages (Matlab, R, etc.) will be provided.

1. Conclusion

In history of databases, the type of data stored in the databases were relatively simple. In the past few years, however, there has been an increasing need for handling new data types in databases, such as temporal data, spatial data, multimedia data, and geographic data and so on. Another major trend in the last decade has created its own issues, for example, the growth of mobile computers, starting with laptop computers, palmtop computers and pocket organizers. In more recent years, mobile phones have also come with built-in computers. These trends have resulted into the development of new database technologies to handle new data types and applications. In this unit, some of the emerging database technologies have been briefly introduced. We have discussed how databases are used and accessed from Internet, using web technologies, use of mobile databases to allow users widespread and flexible access to data while being mobile and multimedia databases providing support for storage and processing of multimedia information. We have also presented how to deal with geographic information data or spatial data and their applications.

4.6 Tutor Marked Assignment

1. What is a mobile database? Explain the architecture of mobile database
2. What is spatial data model? Enumerate the features of spatial database
3. Differentiate between range queries, neighbour queries and spatial joins
4. Enumerate the requirements and Features of SciDB

4.7 Further Readings/Reference

A. Anjomshoaa and A. Tjoa, "How the cloud computing paradigm could shape the future of enterprise information processing", Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services - iiWAS'11, pp. 7-10, 2011.

Abadi, D.J., Madden, S., Ferreira, M.C.: Integrating compression and execution in column-oriented database systems. In Proc. 2006 SIGMOD Conf., June 27-29, Chicago. IL, USA. ACM, New York (2006).

Ailamaki, A., DeWitt, D., Hill, M., Skounakis, M.: Weaving Relations for High Performance. In Proc. of the 27th Int. Conf. on Very Large Databases, Rome, Italy (2001).

Ailamaki, A., DeWitt, D., Hill, M., Wood, D.A.: DBMSs on a Modern Processor: Where Does Time Go? In Proc. 25th Int. Conf. on Very Large Databases, Edinburgh, Scotland (1999).

Bach Pedersen, Torben; S. Jensen, Christian (December 2001). "Multidimensional Database Technology". Distributed Systems Online: 40–46. ISSN 0018-9162

Borne K, Becla J, Davidson I, Szalay A, and Tyson J, 2008, "The LSST Data Mining Research Agenda," AIP Conf. Proc. 1082, pp 347-351.

Cranshaw J, Cuhadar-Donszelmann T, Gallas E, Hrivnac J, Kenyon M, McGlone H, Malon D, Mambelli M, Nowak M, Viegas F, Vinek E, and Zhang Q, 2010, "Event selection services in ATLAS," J. Phys. Conf. Ser. 219 042007.

Cranshaw J, Goosens L, Malon D, McGlone H, and Viegas F, 2008, "Building a scalable event-level metadata service for ATLAS," J. Phys. Conf. Ser. 119 072012.

Cranshaw J, Malon D. Vaniachine A, Fine V, Lauret J, and Hamill P, 2010, "Petaminer:

Daniel Lemire (December 2007). "Data Warehousing and OLAP-A Research-Oriented Bibliography".

E. Gamma, R. Helm, R. Johnson, and J. Vlissidis. Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

Erik Thomsen. (1997). OLAP Solutions: Building Multidimensional Information Systems, 2nd Edition. John Wiley & Sons. ISBN 978-0-471-14931-6.

https://www.stratoscale.com/blog/dbaas/what-is-database-as-a-service/

Hudec, M.: Fuzzy SQL for statistical databases. In MSIS, Meeting on the Management of Statistical Information Systems. Luxembourg. (2008) [Online]. Available: http://www.unece.org/stats/documents/ece/ces/ge.50/2008/wp.12.e.pdf (January 2009)

Idreos, Stratos; Kersten, Martin L; Manegold, Stefan (2007). Database cracking. Proceedings of CIDR.

International Business Machines, Corp. DB2 Spatial Extender – User's Guide and Reference, Version 8.1, 2002.

International Business Machines, Corp. Informix Spatial DataBlade, Version 8.11, 2001.

ISO/DIS 19107:2002. Geographic Information - Spatial Schema, 2002.

ISO/DIS 19111:2002. Geographic Information - Spatial Referencing by Coordinates, 2002.

ISO/IEC 9075-2:1999. Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation), 1999.

ISO/IEC 9075-2:2001 WD. Information Technology – Database Languages – SQL – Part 7: Temporal (SQL/Foundation), 2001.

ISO/IEC 9075-9:2000. Information Technology – Database Languages – SQL – Part 9: SQL/MED, 2000.

Ling Liu and Tamer M. Özsu (Eds.) (2009). "Encyclopedia of Database Systems, 4100 p. 60 illus. ISBN 978-0-387-49616-0.

Malon D, Cranshaw J, and Karr K, 2006, "A flexible, distributed event-level metadata system for ATLAS," Computing in High Energy and Nuclear Physics, Mumbai, India.

Malon D, Van Gemmeren P, Nowak M, and Schaffer A, 2006, "Schema evolution and the ATLAS event store," Computing in High Energy and Nuclear Physics, Mumbai, India.

Mark Tatum, Dan Brennan, Darryl McGowan (2017). Oracle White Paper: *Big Data, Fast Data, All Data*

Modi, A (2017). "Live migration of virtual machines with their local persistent storage in a data intensive cloud". International Journal of High Performance Computing and Networking.

Moghaddam, B., & Pentland, A. (1999). Bayesian image retrieval in biometric databases. In Proceedings of the IEEE International Conference on Multimedia Computing and Systems (Vol. 2, p. 610).

Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications (PDF). Ph.D. Thesis. Universiteit van Amsterdam. May 2002.

Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications (PDF). Ph.D. Thesis. Universiteit van Amsterdam. May 2002.

MonetDB July 2015 Released". https://www.monetdb.org/blog/monetdb-jul2015-released. 31 August 2015.

MonetDB Oct2014 Release Notes". https://www.monetdb.org/OldReleaseNotes/Oct2014. 12 December 2014.

Oracle (2004). Oracle Corporation. *www.oracle.com* OWL (2004). Web Ontology Language. *www.w3.org/ 2001/sw/WebOnt/*

MODULE 4: EMERGING DATABASE MODELS, TECHNOLOGIES AND APPLICATIONS

Unit 4. DATABASE SERVICES AND SERVICE PROVIDERS

4.4.1 Introduction

Database as a service (DBaaS) is the process of application owners paying an outside provider that launches and maintains a cloud database for storage, as opposed to having the application owners control the database themselves. Payments are per-usage and application owners can access their application data as they please. These databases provide the same functionality as a standard relational or non-relational database. DBaaS is beneficial for companies that are trying to avoid the work of configuring, maintaining, and upgrading their own databases. DBaaS lives in the overall realm of software as a service (SaaS), similar to platform as a service (PaaS) and infrastructure as a service (IaaS), where all products are hosted as a service. Other types of databases include relational database tools, NoSQL database tools, graph database tools and more. Developers on a budget also have options with free database software.

4.4.1.2 Objectives:

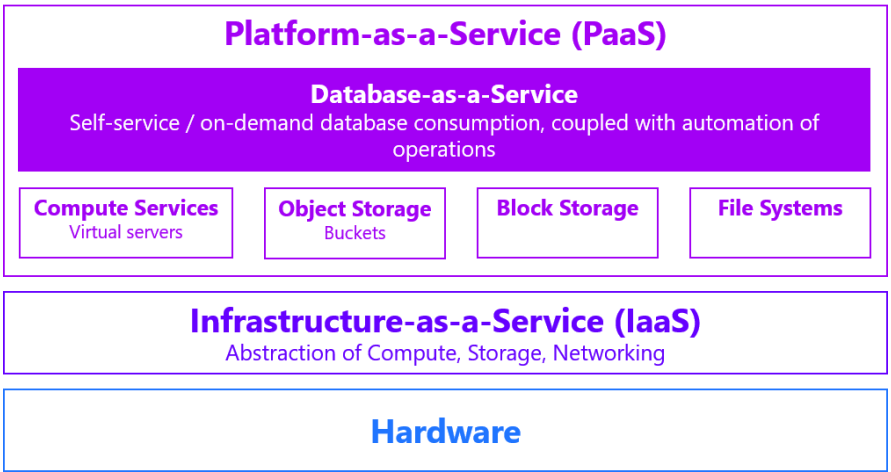
- To understand the concept of Database as a Service
- To understand the set up and operation of DBaaS
- To understand the benefits of DBaaS

4.4.1.3 Database as a Service (DBaas)

(DBaaS) refers to software that enables users to setup, operate and scale databases using a common set of abstractions (primitives), without having to either know or care about the exact implementations of those abstractions for the specific database. The process is as follows:

1. ***Setup :*** Setting up a database involves provisioning a VM on which to run it, installing the database, and configuring it according to a set of parameters. Information Technology (IT) administrators managing the platform can choose to setup databases for their consumers, or enable a self-service model in which developers and DevOps create databases either through an enterprise portal, an SDK, or even using automation tools like Terraform. The self-service model has the advantage of zero IT intervention, freeing up IT admins for more important tasks. Using DBaaS, the time required to setup a database can be reduced from weeks to minutes.
2. ***Operate:*** Once a database has been setup, the platform is responsible for all the back-end operations to maintain it in good health. These include configuration management, automating backups (and enabling easy restore when needed), patches and upgrades, DR, service monitoring (both for the database and the underlying infrastructure) and more. All of these capabilities are provided to the IT administrator as easy single-click operations rather than the complex procedures they would have been without a DBaaS platform.

3. **Scale:** To accommodate increased usage of an application as it evolves and matures, the platform should automatically scale up database instances as needed according to a set of policies. For example, as usage grows beyond a certain threshold, data from a master instance can be automatically distributed to one or more read replica instances. Once data has been distributed over multiple instances, one of the read replicas can also be used for failover.
4. **DBaaS on IaaS:** DBaaS is often delivered as a component of a more



comprehensive platform, which may provide additional services such as Infrastructure-as-a-Service (IaaS). The DBaaS solution would request resources from the underlying IaaS, which would automatically manage the provisioning *compute, storage and networking as needed essentially removing the need for IT to be involved.*

Figure 4.4.1: DBaaS on IaaS

Source: https://www.stratoscale.com/blog/dbaas/what-is-database-as-a-service/

4.4.1.4 Who Users DBaaS:

It is important to understand that like other cloud technologies, DBaaS has two primary consumers:

1. The IT organization which manages and maintains the cloud
2. The end user who consumes the cloud resources, typically, developers and DevOps.

The IT organization deploys the DBaaS solution enabling end users (developers and DevOps) to provision a database of their choice, on-demand, from a catalog of supported databases, which could include both relational and non-relational databases. The IT organization can configure the DBaaS to support specific releases of these software titles, and can further restrict the configurations that specific users can provision. For example, developers may only be allowed to provision databases with a small memory footprint using traditional disks while DevOps could provision higher capacity servers with SSD's. Finally, the IT organization can setup policies for standard database operations like backups, DR and security policies to ensure that the data is properly saved from time to time to allow for recovery when required.

Typically, an end user would access the DBaaS system through a portal that offers a selection of different database titles, and in a variety of different configuration options. With a few clicks, the user specifies the required database and its corresponding

configuration for provisioning. The DBaaS system quickly provisions the database and returns a query-able endpoint such as: *mysql://192.168.15.243:3306/* . The user can then use this directly in an application. The DBaaS system provides simple mechanisms to add users, create schemas and grant permissions to different users as required by the application.

4.4.2 Benefits of DBaaS

A DBaaS solution provides an organization a number of benefits, the main ones being:

1. Developer agility
2. IT productivity
3. Application reliability and performance
4. Application security

4.4.3 Developer agility

Deploying a database is a multi-step process including the provisioning of compute, storage and networking components, configuring them properly and installing the database software. In most enterprises, this process must go through the organization's IT department and is typically, something like the following:

1. Developer opens a request in the IT ticketing system
2. Ticket sits in the queue until it gets to the top of the list according to IT priorities
3. IT evaluates ticket, and if the request is approved goes about allocating the required compute, storage and networking resources needed for the developer's database (in some cases, each of these is also handled by a separate sub-department which has its own ticketing system and set of priorities)

1. IT configures the allocated resource
2. IT installs and configures the database to utilize the underlying infrastructure according to its internal policies

3. IT provides the developer with an entry point to the database and the developer takes it from there.

Not only is this process prone to errors and omissions, it is also extremely time-consuming. The above series of actions that might be completed in hours or days, can be stretched to days, weeks and even months. This is unacceptable for developers who are trying to shorten development cycles and release faster. A DBaaS solution radically improves provisioning time by automating the process described above. The IT organization establishes the standards by which databases will be provisioned and configures the DBaaS accordingly. Once database provisioning is standardized, and the DBaaS is configured, deploying a database is a task that can be handed back to developers who can now provision databases for themselves without requiring any intervention by IT through a simple API call or a few clicks on a UI portal. This not only gives developers the agility they need by enabling them to provision databases in a "single-click self-service" model, but also, databases will always be provisioned in a consistent manner that is aligned with the best practices for that particular database.

4.4.3 **IT productivity**

IT is responsible for the day-two operations of the enterprise's databases including things like tuning, configuration, monitoring, patching, upgrading, resizing periodic backups, and so on; all the things that must be done to keep databases in proper working order. As enterprises grow, and with them, the number and types of databases that must be managed and maintained, IT resources get stretched very thin (explaining the long lead time that developers have to wait before IT provisions them with a database).

DBaaS solutions enable IT to manage a much larger number of databases by treating them as "cattle" rather than "pets". By providing abstractions for and automating tasks involved with day-two operations, a DBaaS solution vastly simplifies IT's job, allowing operations

like upgrades and configuration changes to be done on a fleet of databases with a single action. Once relieved from micro-managing all of an enterprise's databases, IT can focus more on activities like establishing the standards of operation for the enterprise and providing quicker service for the developers who they serve.

4.4.3.1 Application reliability and performance

Modern DBaaS solutions make it easy to keep your databases highly available and running at peak performance. Through support for read replicas, in the event of a failure, the system automatically reroutes traffic to a replica ensuring system availability at all times. The system also monitors your databases to identify increased demand on resources. Using scaling policies based on resource usage thresholds, you can configure the system to automatically scale out by provisioning additional resources as demand increases, and then scale back in once demand is reduced releasing resources for other applications.

4.4.3.2 Application security

Many database engines natively provide security features such as data encryption both at rest and in transit, each using its own data structures and APIs. A DBaaS solution provides consistent management of security for all the different types of databases you might use in your organization, while adding some security features of its own. In addition to native data encryption, you might look for things like end-to-end network security with micro-segmentation, virtual private networks and security groups. A DBaaS solution might also integrate with common enterprise user stores such as LDAP and Active Directory for user authentication, and then apply fine-grained access control via different permission policies.

4.4.3.3 Some DBaaS Providers:

1. **Amazon Aurora:** MySQL and PostgreSQL-compatible relational database engine that combines the speed and availability of high-end commercial databases with the simplicity and cost-effectiveness of open source databases.
2. **IBM Db2:** Makes it easy to deploy data wherever it's needed, fluidly adapting to changing needs and integrating with multiple platforms, languages and workloads. Supported across Linux, Unix, and Windows operating systems.
3. **Amazon Relational Database Service (RDS):** A web service that makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks. Amazon RDS gives access to the capabilities of a familiar MySQL, Oracle or Microsoft SQL Server database engine. This means that the code, applications, and tools already used today with existing databases can be used with Amazon RDS. Amazon RDS automatically patches the database software and undertakes regular back ups, storing the backups for a user-defined retention period and enabling point-in-time recovery. You benefit from the flexibility of being able to scale the compute resources or storage capacity associated with your relational database instance via a single API call. In addition, Amazon RDS makes it easy to use replication to enhance availability and reliability for production databases. Amazon RDS for MySQL also enables you to scale out beyond the capacity of a single database deployment for read-heavy database workloads. As with all Amazon Web Services, there are no up-front investments required, and you pay only for the resources you use.
4. **MongoDB Atlas:** Cloud-hosted MongoDB service engineered and run by the same team that builds the database. It incorporates operational best practices we've learned from optimizing thousands of deployments across startups and the Fortune 100. Build on MongoDB Atlas with confidence, knowing you no longer need to

worry about database management, setup and configuration, software patching, monitoring, backups, or operating a reliable, distributed database cluster.

5. **Ninox Database:** Lets you integrate everything you need - applications from different departments to streamline your operations - e.g. CRM, Sales, ERP, Projects, HR & Administration.
6. **Azure SQL Database:** A relational database-as-a service using the Microsoft SQL Server Engine. SQL Database is a high-performance, reliable, and secure database you can use to build data-driven applications and websites in the programming language of your choice, without needing to manage infrastructure.
7. **Aiven:** Provides managed cloud service hosting for software infrastructure services. Acting as a central hub for all database needs, relational and non-relational database services along with a visualization suite and high-throughput message broker are offered. Service offerings includes Kafka, PostgreSQL, MySQL, Elasticsearch, Cassandra, Redis, InfluxDBand Grafana. All Aiven services are billed by the hour based on actual usage with no hidden fees.
8. **Kintone:** A no-code business application platform that allows non-technical users to create powerful apps, workflows, and databases for their teams and organizations. Using clicks instead of code, kintone users can build apps that automate business processes, collaborate on projects/tasks, and quickly report on complex data. For business users that need to get started right away, kintone also provides dozens of pre-built applications for a variety of use cases such as CRM, project management, inventory management, and much more.
9. **Google Cloud BigTable:** Google's NoSQL Big Data database service. It's the same database that powers many core Google services, including Search, Analytics, Maps, and Gmail. Bigtable is designed to handle massive workloads at consistent low latency and high throughput, so it's a great choice for both operational and analytical applications, including IoT, user analytics, and financial data analysis.

10. **IBM Cloudant:** Fully Managed — IBM Cloud service provides a fully managed, distributed JSON document database. Instantly deploy an instance, create databases, and independently scale throughput capacity and data storage to meet your application requirements. IBM expertise takes away the pain of hardware and software provisioning, patching and upgrades, while offering a 99.95 percent SLA. Secure — Cloudant is ISO27001, SOC 2 Type 2 compliant and HIPAA ready. All data is encrypted over the wire and at rest with optional user-defined key management through IBM Key Protect.
11. **Symas LMDB:** A memory-mapped DB with read performance of an in-memory database and persistence of standard disk-based databases.
12. **Vertabelo:** Vertabelo enables users to create database model, share it with team, and finally generate SQL scripts instead of writing them manually.
13. **Webair Database-as-a-Service (DBaaS):** Webair provides Database-as-a-Service (DBaaS), a reliable and secure database management solution that gives your business simple, efficient and always available access to its mission-critical data.
14. **mLab:** A fully managed cloud database service featuring automated provisioning and scaling of MongoDB databases.
15. **Alibaba Cloud ApsaraDB:** Alibaba Cloud develops highly scalable cloud computing and data management services.

1. Big Data Processing and Distribution Software

Big Data processing and distribution systems offer a way to collect, distribute, store, and manage massive, unstructured data sets in real time. These solutions provide a simple way to process and distribute data amongst parallel computing clusters in an organized fashion. Built for scale, these products are created to run on hundreds or thousands of machines simultaneously, each providing local computation and storage capabilities. Big data processing and distribution systems provide a level of simplicity to the common business

problem of data collection at a massive scale and are most often used by companies that need to organize an exorbitant amount of data. Many of these products offer a distribution that runs on top of the open-source big data clustering tool Hadoop.

4.4.4.1 Big Data Processing Requirements:

We can classify Big Data requirements based on its five main characteristics:

1. Volume:

1. Size of data to be processed is large—it needs to be broken into manageable chunks.
2. Data needs to be processed in parallel across multiple systems.
3. Data needs to be processed across several program modules simultaneously.
4. Data needs to be processed once and processed to completion due to volumes.
5. Data needs to be processed from any point of failure, since it is extremely large to restart the process from the beginning.

6. Velocity:

1. Data needs to be processed at streaming speeds during data collection.
2. Data needs to be processed for multiple acquisition points.
3. Variety:

1. Data of different formats needs to be processed.
2. Data of different types needs to be processed.
3. Data of different structures needs to be processed.
4. Data from different regions needs to be processed.

1. Ambiguity:

1. **Big Data** is ambiguous by nature due to the lack of relevant metadata and context in many cases. An example is the use of M and F in a sentence—it can mean, respectively, Monday and Friday, male and female, or mother and father.

2. **Big Data** that is within the corporation also exhibits this ambiguity to a lesser degree. For example, employment agreements have standard and custom sections and the latter is ambiguous without the right context.

1. Complexity:

1. Big Data complexity needs to use many algorithms to process data quickly and efficiently.
2. Several types of data need multipass processing and scalability is extremely important.
3. Processing large-scale data requires an extremely high-performance computing environment that can be managed with the greatest ease and can performance tune with linear scalability.

4.4.4.2 Big Data Processing and Distribution Software:

To qualify for inclusion in the Big Data Processing and Distribution category, a product must:

1. Collect and process big data sets in real-time
2. Distribute data across parallel computing clusters
3. Organize the data in such a manner that it can be managed by system administrators and pulled for analysis
4. Allow businesses to scale machines to the number necessary to store its data

Some available software include:

1. **Google BigQuery:** Google's fully managed, petabyte scale, low cost enterprise data warehouse for analytics. BigQuery is serverless. There is no infrastructure to manage and you don't need a database administrator, so you can focus on analyzing data to find meaningful insights using familiar SQL. BigQuery is a powerful Big Data analytics platform used by all types of organizations, from startups to Fortune 500 companies.

2. **Snowflake**: Cloud data platform shatters the barriers that have prevented organizations of all sizes from unleashing the true value from their data. Thousands of customers deploy Snowflake to advance their organizations beyond what was possible by deriving all the insights from all their data by all their business users. Snowflake equips organizations with a single, integrated platform that offers the only data warehouse built for the cloud; instant, secure, and governed access to their entire network of data; and a core architecture to enable many types of data workloads, including a single platform for developing modern data applications.
3. **Amazon EMR:** A web-based service that simplifies big data processing, providing a managed Hadoop framework that makes it easy, fast, and cost-effective to distribute and process vast amounts of data across dynamically scalable Amazon EC2 instances.
4. **Qubole:** Qubole is revolutionizing the way companies activate their data--the process of putting data into active use across their organizations. With Qubole's cloud-native Data Platform for analytics and machine learning, companies exponentially activate petabytes of data faster, for everyone and any use case, while continuously lowering costs. Qubole overcomes the challenges of expanding users, use cases, and variety and volume of data while constrained by limited budgets and a global shortage of big data skills. Qubole's intelligent automation and self-service supercharge productivity, while workload-aware auto-scaling and real-time spot buying drive down compute costs dramatically. Qubole offers the only platform that delivers freedom of choice, eliminating legacy lock in--use any engine, any tool, and any cloud to match your company's needs.
5. **Azure HDInsight:** HDInsight is a fully-managed cloud Hadoop offering that provides optimized open source analytic clusters for Spark, Hive, MapReduce, HBase, Storm, Kafka, and R Server.

6. **Google Cloud Dataflow:** Fully-managed service for transforming and enriching data in stream (real time) and batch (historical) modes with equal reliability and expressiveness. And with its serverless approach to resource provisioning and management, you have access to virtually limitless capacity to solve your biggest data processing challenges, while paying only for what you use.
7. **Snowplow Analytics:** Best-in-class data collection platform built for Data Teams. With Snowplow you can collect rich, high-quality event data from all your platforms and products. Your data is available in real-time and is delivered to your data warehouse of choice where it can easily be joined with other data sets and used to power BI tools, custom reports or machine learning models. The Snowplow pipeline runs in your cloud account (AWS and/or GCP), giving you complete ownership and control of your data. Snowplow frees you to ask and answer any questions relevant to your business and use case, using your preferred tools and technologies.
8. **Oracle Big Data Cloud Service:** offers an integrated portfolio of products to help organize and analyze diverse data sources alongside existing data.
9. **Cloudera:** Delivers an enterprise data cloud for any data, anywhere, from the Edge to AI. Enables people to transform vast amounts of complex data into clear and actionable insights to enhance their businesses and exceed their expectations. Cloudera is leading hospitals to better cancer cures, securing financial institutions against fraud and cyber-crime, and helping humans arrive on Mars — and beyond. Powered by the relentless innovation of the open-source community, Cloudera advances digital transformation for the world's largest enterprises.
10. **Alibaba MaxCompute:** A general purpose, fully managed, multi-tenancy data processing platform for large-scale data warehousing. MaxCompute supports various data importing solutions and distributed computing models, enabling users

to effectively query massive datasets, reduce production costs, and ensure data security.

Companies commonly have a dedicated administrator for managing big data clusters. The role requires in-depth knowledge of database administration, data extraction, and writing host system scripting languages. Administrator responsibilities often include implementation of data storage, performance upkeep, maintenance, security, and pulling the data sets. Businesses often use big data analytics tools to then prepare, manipulate, and model the data collected by these systems.

4.4.5 Big Data Analytics with Oracle

Oracle's technology provides a solution that is a fully complete analytic environment that supports full-spectrum data ingest, wrangling, data exploration & discovery through advanced and predictive analytics. It represents the combination of software, cloud computing and/or supporting hardware that has been professionally engineered, optimized, developed and deployed to support the analytic challenges faced today. A key differentiated objective is to empower analysts to explore, test, and evaluate in a self-service fashion thus reducing the need for costly programmers and data scientists.

Oracle has created a holistic, standards-based and unified approach to provide integrated analysis for all data types, analytic methods and user classes.

FAST DATA

1. With exploding intelligence data from increased number of connected devices, cyber sensors, collection platforms and social networks, there is an increase in the volumes and speed of dynamically changing data. High-velocity data brings high value, especially to national security decision-making processes. However, some of this data loses its operational value in a short time frame. Big Data allows the luxury of

time in processing for actionable insight. Fast Data, on the other hand, requires extracting the maximum value from highly dynamic and strategic intelligence. It requires processing much faster and facilitates taking timely action as close to the generated data as possible. Fast Data can also be a means of getting early answers or tip-offs of what's in your big data holdings. Industry and government needs have led to huge internal research investments at Oracle. Oracle has responded with a robust platform for Fast Data and Stream Analytics.

4.4.4.1 Oracle Stream Analytics

The Oracle Stream Analytics platform provides a compelling combination of: an easy-to-use visual façade to rapidly create and modify Event Stream Processing applications, an advanced streaming analytical capabilities, and a comprehensive, flexible and diverse runtime platform that uses Apache Spark, Kafka and SOA streaming technology. Oracle Stream Analytics provides a pre-developed library of algorithms and detection features to leverage spatial, statistical, machine learning and well-known patterns. The abstraction and definition of streams and targets allow immediate joining of streaming data to gold copy look-up databases, and explorations that provide a stunning visual representation of real time event data.

The user has the ability to rapidly create stream exploration and analysis without writing custom code in Apache Storm. In addition to real-time event sourcing, the Oracle Stream Analytics design environment and runtime execution supports standards-based, **continuous query execution** across both event streams and persistent data stores. This enables the platform to act as the heart of intelligence for systems needing answers in microseconds to discern patterns and trends that would otherwise go unnoticed. Event processing use cases require the speed of in-memory processing with the mathematical accuracy and reliability of standard database SQL. This platform listens to incoming event

streams and executes registered queries continuously, in-memory on each event, utilizing advanced, automated algorithms for query optimization. Examples of this type of detection:

1. Correlated events: If event A happens, event B almost always follows within 2 seconds of it.
2. Missing or Out-of-Sequence events: Events A, B, C should occur in order. C is seen immediately after A, without B would be an alert.
3. Spatial/Temporal events: Target of a particular type has passed through a geo-boundary at a particular time while an attribute about the target registers positive in a watch list database.

The Oracle Stream Analytics platform also allows for both SQL and Java code to be combined to deliver robust event processing applications, leveraging standard industry terminology to describe event sources, processes, and event output or syncs. The platform provides a meta-data driven approach to defining and manipulating events within an application. Developers and analysts use a visual, directed-graph canvas and palette for application design to quickly outline the flow of events and processing across both streams and data sources. Developing the flow through drag and drop modelling and configuration wizards, the analyst can then enter the appropriate metadata definitions to connect design to implementation.

4.4.6 DATA PLATFORM

The data platform is the layer of the data management and analytic architecture where data is staged, transformed, secured and managed. The platform must provide the scalability, security and high availability that enterprises require. Oracle has focused significant R&D and innovation in this area because we realize that it is critical to all other aspects of analytics, cyber protection and mission outcomes. Oracle believes that the data platform is a unified environment, which may include an HDFS data lake, SQL and NoSQL databases,

with data transformation, data cleansing, query, and movement functions that we call the factory. All of these capabilities work in unison to bring a modern data platform to the enterprise. Where data resides, where it's staged, where it is stored, where it is managed within the platform is really dependent on numerous factors including: type, security, analysis, governance, compliance, cost and processing method. Choosing one data method, like SQL or NoSQL, for every use case is not wise. Likewise, using Hadoop/HDFS for everything is not wise. All three serve specific purposes. Oracle has invested significantly making this data platform unified. Below outlines the technologies that make up the data platform eco-system.

4.4.6.1 **Gold Data Using ORACLE 12c**

The gold data concept is often referred to as "gold" because of the value. Gold data is actionable. It is data that has been cleansed and is as accurate as possible. The gold data environment must have several critical features: encryption, security, high availability, survivability and transactional integrity and high performance. The platform must also be able to handle all data types such as: spatial, graph, xml, json/key value, video, imagery and relational.

All too often enterprises put in place separate databases for every data type. This causes analytical, management and security turmoil because there are separate databases for graph, spatial, relational, xml, JSON etc. Oracle 12c is a multi-model database. It is capable of providing gold data services for all of these data types in the same database. This improves analytics by creating cohesive result sets, improves security by securing all of the data together and significantly reduces the cost of all the software, infrastructure and labor compared to a multiple database gold environment.

4.4.6.2 Oracle NoSQL Database

The Oracle NoSQL Database provides network-accessible multi-terabyte distributed key/value pair storage with predictable latency. Data is stored in a very flexible key-value format, where the key consists of the combination of a major and minor key (represented as a string) and an associated value (represented as a JSON data format or opaque set of bytes). It offers full Create, Read, Update and Delete (CRUD) operations, with adjustable durability and consistency guarantees. It also provides a powerful and flexible transactional model (with ACID) that eases application development.

1. The Oracle NoSQL Database is designed to be a highly available and extreme scalable system, with predictable levels of throughput and latency, while requiring minimal administrative interaction.
2. It is also network topology and latency aware. The database driver working in conjunction with highly scalable, fault tolerant, high throughput storage engine enables a more granular distribution of resources and processing, which reduces the incidence of hot spots and provides greater performance on commodity-based hardware.

4.4.6.3 Big Data SQL

Big data's great potential has been touted for years. For many organizations, however, the challenge to making their big data vision a reality is that their ability to collect data has gotten far ahead of their ability to use it.

One of the hurdles is that an enterprise's data is typically scattered across departments, systems, and regions, and much of what is collected is of low or unknown value. Vast quantities of this data are now stored in Apache Hadoop, which makes it possible to store and process huge amounts of data at low cost. Other data is housed in relational and NoSQL databases.

The holy grail of a big data strategy is to be able to easily leverage the information in all of these data stores to get a deeper, richer, and far more valuable view of customers, business processes, and opportunities. But Hadoop programmers are in short supply, so many organizations don't have the skills they need to leverage Hadoop data. And it's not feasible to move all of that data to a single data store; the cost and the security issues are prohibitive.

Oracle Big Data SQL enables organizations to get the utmost value from data by providing queries to big data systems integrated with existing enterprise information architectures. You can then quickly leverage big data into reports or applications using existing interfaces.

In addition to simplified access and integration, Oracle Big Data SQL uses some of the proven high performance technology of Oracle Exadata and the industry-leading security features of Oracle Database to provide super-fast speed and enterprise-class security across all of your data stores.

Leveraging Oracle Exadata Smart Scan technology, Oracle's Smart Scan on Hadoop processes SQL queries at the Hadoop storage level where data is located, scans the data, and brings back only the relevant data to the end user. Less data moved means faster results—and speed is critical when leveraging big data for real-time innovations in national security processes.

On the security front, Oracle Big Data SQL extends the advanced security capabilities of the Oracle database to Hadoop and NoSQL data. With Oracle Big Data SQL, you can take advantage of proven Oracle Database security solutions for data redaction and privilege analysis, with strong controls that limit privileged user access to data.

4.4.6.4 Oracle Advanced Analytics and Machine Learning

Oracle provides several enabling technologies for data analytics, statistical analysis, time-series analysis, modelling, and machine learning requirements. These technologies can actually be used in both the product data platform and the data lab. Traditional advanced analytics are inherently weak at information technology management such as:

1. data extracts and data movement
2. data duplication resulting in no single-source of truth
3. data security exposures
4. multiple analytical tools (commercial and open source) and languages (SAS, R, SQL, Python, SPSS, etc.)

Problems become particularly egregious during a deployment phase when the worlds of data analysis and information management collide.

1. In-Database Processing with Oracle Advanced Analytics

Oracle Advanced Analytics extends the database into a comprehensive advanced analytics platform for big data analytics. With Oracle, powerful analytics are performed directly on data in the database. Results, insights, and realtime predictive models are available and managed by the database.

A data mining model is a schema object in the database, built via a PL/SQL API that prepares the data, learns the hidden patterns to build an OAA model which can then be scored via built-in OAA data mining SQL functions. When building models, Oracle Advanced Analytics leverages existing scalable technology (e.g., parallel execution, bitmap indexes, aggregation techniques) and additional developed new Oracle Advanced Analytics and Oracle Database technologies (e.g., recursion within the parallel infrastructure, IEEE float, automatic data preparation for binning, handling missing values, support for unstructured data i.e. text, etc.). The true power of embedding data mining functions within the database as SQL functions is most evident when scoring data mining

models. Once the models have been built by learning the hidden patterns in the historical data, applying the models to new data inside the database is blazingly fast. Scoring is then just a row-wise function. Hence, Oracle Advanced Analytics can "score" many millions of records in seconds and is designed to support online transactional processing (OLTP) environments.

Using Exadata's smart scan technology it gets better. With Oracle Advanced Analytics running in an Exadata environment, SQL predicates and OAA predictive models get pushed down to the hardware layer for execution.

1. For Oracle Exadata environments, pushed to Exadata storage level for execution
2. For Oracle Big Data Appliance (Hadoop) environments, pushed to BDA storage level for execution.

The Oracle approach to data analytics is to create a platform that is open, feature rich, integrated, standards-based, and performant. It makes big data simple by integratingHadoop, NoSQL and SQL across the Fast Data, Data Factory, Data Lake and the Data Lab environments. The ultimate goal is to provide a cost efficient, easy to use platform that transforms all data across the enterprise into actionable information that leads to better decisions. We are proud that 3rd party analysts agree with our innovations, as Gartner has chosen Oracle as the leader in Gartner's Magic Quadrant for data platforms for data analytics.

4.4.7 Non-native Database Management Systems

Non-native database management systems allow users of outside applications to interact with a database by retrieving data with specific query languages. Non-native database management systems provide a platform for building queries to access and administrate necessary data.

4.4.7.1 Features of Non-native Database Management Systems

To qualify for inclusion in the Non-Native Database Management System category, a product must:

1. Provide access to information stored in a database using query language
2. Offer a platform for building and executing queries

Some features include:

1. Connection Manager - Allow users to connect natively to the vendor's database whether on-premise or DBaaS.
2. Browser - Allow users to browse all the different database/schema objects and their properties effective management.
3. Editor - A way to create and maintain scripts and database code with debugging and integration with source control.
4. Unit Testing (Oracle) - Ensures code is functionally tested before it is released into production.
5. Static code review (Oracle) - Ensures code meets required quality level using a rules-based system.
6. SQL Optimization - Provides developers with a way to tune and optimize SQL statements and database code without relying on a DBA. Advanced optimization enables DBAs to tune SQL effectively in production.
7. Scalability testing and database workload replay - Ensures that database code and SQL will scale properly before it gets released into production.

4.4.8 Some Non-native Database Management Systems

1. **Toad For Oracle**: Toad solutions are desktop tools that give DBAs, developers and analysts a proactive, automated approach to developing and managing databases, so organizations can spend more time on strategic initiatives and less time on mundane, repetitive tasks. Quest offers several Toad editions, which support

databases such as Oracle, SQL Server, DB2, MySQL, SAP Solutions, and more, as well as offerings built specifically for database developers, DBAs and business analysts. Used by millions, Toad for Oracle is the flagship brand in the Toad portfolio, and the leading database development and optimization toolset on the market. Toad Data Point is an analyst toolset, which connects to nearly any data source and is purpose-built for an organization's data provisioning and reporting requirements. Toad solutions help organizations maximize their investments in data and database technologies by empowering database professionals to automate processes, minimize risks and cut project timelines in half.

2. **phpMyAdmin:** A free software tool written in PHP, intended to handle the administration of MySQL over the World Wide Web.
3. **Amazon Athena:** an interactive query service designed to make it easy to analyze data in Amazon S3 using standard SQL.
4. **SQLyog:** Helps developers and database administrators (DBAs) to create and manage MySQL databases with ease. SQLyog is the most complete & easy MySQL GUI tool that helps save time & increase productivity through numerous powerful features such as autocomplete, query building, query profiler, Visual Schema designer and much more.
5. **Navicat Premium:** A database administration tool that allows you to simultaneously connect to MySQL, MariaDB, SQL Server, Oracle, PostgreSQL, and SQLite databases from a single application. Navicat Premium combines functions of other Navicat products and supports most of the features used in modern database management systems, such as Stored Procedures, Events, Triggers, Functions, Views, etc.
6. **DBeaver:** A database management tool that suppor various databases including MySQL and PostgreSQL.

7. **DbVisualizer:** The universal database tool for developers, DBAs and analysts. It is the perfect solution since the same tool can be used on all major operating systems accessing a wide range of databases.
8. **TablePlus:** A native tool with elegant UI that allows you to simultaneously manage multiple databases such as MySQL, Postgres, SQL Server, SQLite, Microsoft SQL Server and more.
9. **Ingres:** Provides easy migration from MySQL and proprietary databases such as Oracle, SQL Server and Sybase.
10. **DBHawk:** A web-based database management and data security tool. DBHawk is compatible with various databases, including on-premises as well as databases hosted in the cloud. DBHawk provides central data security with 2FA, LDAP integration, Okta integration, data logging and auditing, and object access control. DBHawk is supported with Oracle, DB2, SQL Server, MySQL, PostgreSQL, Amazon RDS, Amazon Aurora, Azure SQL, Amazon Redshift, Greenplum, Netezza, Teradata, SAP-Hana, MariaDB, MongoDB, Hadoop, Amazon Athena (S3), and Cassandra.

These systems can be utilized by database administrators, programmers, and other employees searching for information stored in the database. Query languages often cater to specific types of databases, including DBaaS, NoSQL database tools, relational database tools and more.

4.4.9 Conclusion

Most enterprises today operate applications that require several different database technologies, a departure from recent years where the ‘corporate standard’ mandated a single database solution for all application needs. Database-as-a-Service provides a framework within which enterprises can operate all these different databases. It provides

end users with improved agility through simplified provisioning and operation, and the flexibility to choose from a number of pre-configured options established by the IT organization. DBaaS also improves the operation of fleets of diverse databases through automation and standardization allowing IT organizations to cost-effectively offer their users a number of database choices while also ensuring that these databases are operated in a safe and secure way and in compliance with established best practices.

4.4.10 Tutor Marked Assignment

1. What is big data analytics? State the requirements for big data processing
2. What are the features of Non-native Database Management Systems?
3. What a mobile database? Describe the characteristics of Mobile Database
4. List the categories three typical types of spatial queries and enumerate the features of spatial databases

4.4.11 References and Further Readings

Atzeni, P., Ceri, S., Paraboschi, S., & Torlone, R. (1999). *Database systems: concepts, languages & architectures* (Vol. 1). London: McGraw-Hill.

Batini, C., Ceri, S., & Navathe, S. B. (1992). *Conceptual database design: an Entity-relationship approach* (Vol. 116). Redwood City, CA: Benjamin/Cummings.

Connolly, T. M., & Begg, C. E. (2005). *Database systems: a practical approach to design, implementation, and management*. Pearson Education.

Elmasri, R., & Navathe, S. B. (2011). *Database systems* (Vol. 9). Boston, MA: Pearson Education.

Singh, S. K. (2011). *Database systems: Concepts, design and applications*. Pearson Education India.

MODULE 4 – EMERGING DATABASE MODELS, TECHNOLOGIES AND APPLICATIONS

Unit 5: MODERN DATABASE APPLICATIONS

4.5.1 Introduction:

Most of our traditional tools for formal modelling, reasoning and computing are crisp, deterministic and precise in nature. Precision assumes that the parameters of a model represent exactly either our perception of the phenomenon modelled or the features of the real system that has been modelled. Certainty eventually indicates that we assume the structures and parameters of the model to be definitely known. However, if the model or theory asserts factuality, then the modelling language has to be suited to model the characteristics of the situation under study appropriately. However we have a problem. For factual models or modelling languages, two major complications arise:

1. **Real situations** are very often not crisp and deterministic and cannot be described precisely i.e. real situations are very often uncertain or vague in a number of ways.
2. **Complete description** of a real system would require far more detailed data than a human being could ever recognize and process simultaneously.

Hence, among the various paradigmatic changes in science and mathematics in last century, one such has been the concern of the concept of uncertainty. In science this change is manifested by a gradual transition, from a view, which stated that uncertainty is undesirable to an alternative view that accepts uncertainty as an integral part of the whole system that is essential to model the real world.

4.5.1.2 Objectives:

- To understand the limitations of conventional SQL

- To understand queries based on fuzzy logic
- To understand and describe some of the modern databases

4.5.1.3 Classical SQL and its limitations:

Users search databases in order to obtain data needed for analysis, decision making or to satisfy their curiosity. The SQL is a standard query language for relational databases. The simply SQL query is as follows:

select attribute_1,…,attribute_n
from T
where attribute_p > P and attribute_r < R

The result of the query is shown in graphical mode in figure 4.11. Values P and R delimitthe space of interesting data. Small squares in the graph show database records. In the graph it is obviously shown that three records are very close to meet the query criterion. These records could be potential customers and direct marketing could attract them or municipalities which almost meet the criterion for some financial support for example.

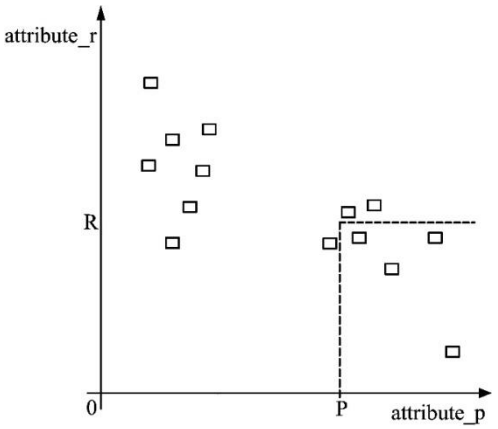


Figure 4.11: The result of the classical query
Source: Miroslav Hudec (2009).

The SQL uses the crisp logic in querying process that causes crisp selection. It means that the record would have not been selected even if it is extremely close to the intent of the query criterion. As the criterion becomes more and more complex, the set of records selected by the WHERE statement becomes more and more crisp.

If the classical SQL is used for solving this problem, the SQL relaxation would have to be used in the following way:

select attribute_1,…,attribute_n
from T
where ateribute_p > P-p and attribute_r < R+r

where p and r are used to expand the initial query criteria to select records that almost meet the query criteria. This approach has two disadvantages. First, the meaning of the initial query is diluted in order to capture adjacent records. The meaning of a query: "where attribute_p is more than P" is changed and adjacent records satisfy a query in the same way as initial ones. More precisely, the difference between original and adjacent data (caught records along the "edge" of interesting space) does not exist. Secondly problem rises from the question: what about records that are very close to satisfy the new expanded query and it is useful to make another expanding of a query. In this way more data from the database is selected, but the user has lost the accuracy of his query.

Many applications have created uncountable accesses to wide variety of data. The data and the classical access to data are simply not enough in many cases. In cases when the user cannot unambiguously separate interesting data from not interesting by sharp boundaries or when the user wants to obtain data that is very close to meet the query criterion and to know the index of distance to full query satisfaction, it is necessary to adapt the SQL to these requirements.

4.5.2. Queries based on Fuzzy Logic

A Query Compatibility Index (QCI) is used to indicate how the selected record satisfies a query criterion. The QCI has values from the [0, 1] interval with the following meaning: 0 - record does not satisfy a query, 1 - record fully satisfies the query, interval (0, 1) - record partially satisfies a query with the distance to the full query satisfaction.

Conditions in queries contain these comparison operators: >, <, =, ≠ and *between* when numerical attributes are used. These crisp logical comparison operators are adapted for fuzzy queries in the following way: operator > (greater than) was improved with fuzzy set "Big value" (figure 4.12a), operator <(less than) was improved with fuzzy set "Small value" (figure 4.12b) and operator = (equal) was improved with fuzzy set "About value" (figure 4.12c). Operator ≠ is the negation of the operator = so this operator is not further analzsed. Analogous statement is valid for the operator *between* because it is similar to the operator = from the fuzzy point of view. Other types of fuzzy sets could be added in the future to catch other linguistic expressions.

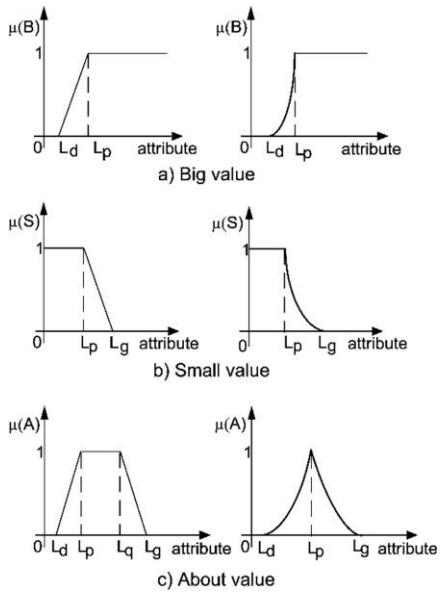


Figure 4.12: Fuzzy Sets
Source: Miroslav Hudec (2009).

To implement this a fuzzy query interpreter, which transforms fuzzy queries to the classical SQL structure, was developed. In this way, queries based on linguistic expressions on client side are supported and are accessing relational databases in the same way as with the classical SQL. Figure 3 shows this model. The first step of querying process (to select records that have QCI>0) is situated in parts 1, 2 and 3. Lower and/or upper limits of linguistic expressions are calculated and converted into SQL query in the part 1. Thus created SQL query selects data from database (part 2) and saves it into the temporary table (part 3). The second step uses data from part 3 for further calculations. Firstly, the chosen analytical form of the fuzzy set (from part 1) is used to determine the membership degree of each selected record to appropriate fuzzy set. Secondly, t-norm and/or t-conorm function also defined in the part 1 are used to calculate the QCI value.

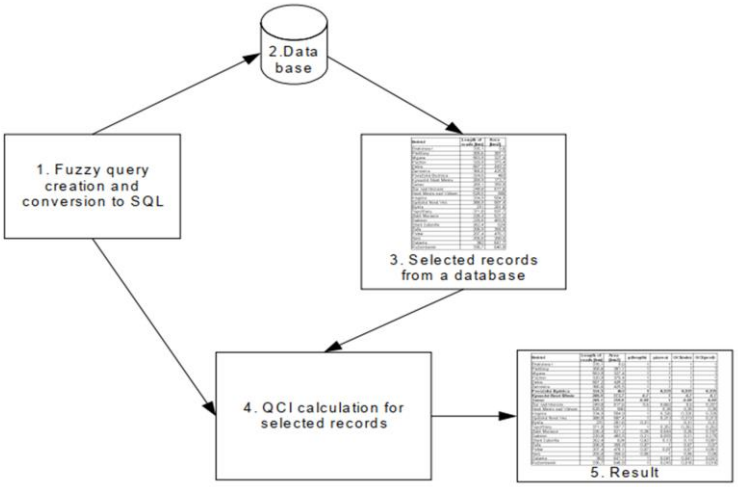


Figure 1.3: Structure of the fuzzy SQL
Source: Miroslav Hudec (2009).

4.5.3 On-line Analytical Processing Databases (OLAP)

OLAP databases are geared toward analyzing data rather than updating data. They are used to drive business processes based on statistical analysis of data and what-if analysis. The main feature of OLAP databases is speed of querying and multi-dimensionality. OLAP tools enable users to analyze multidimensional data interactively from multiple perspectives. OLAP consists of three basic analytical operations: consolidation (roll-up), drill-down, and slicing and dicing. Consolidation involves the aggregation of data that can be accumulated and computed in one or more dimensions. For example, all sales offices are rolled up to the sales department or sales division to anticipate sales trends. By contrast, the drill-down is a technique that allows users to navigate through the details. For instance, users can view the sales by individual products that make up a region's sales. Slicing and dicing is a feature whereby users can take out (slicing) a specific set of data of the OLAP cube and view (dicing) the slices from different viewpoints. These viewpoints are sometimes called dimensions (such as looking at the same sales by salesperson, or by date, or by customer, or by product, or by region, etc.)

4.5.4 Overview of OLAP Systems

At the core of any OLAP system is an OLAP cube (also called a 'multidimensional cube' or a hypercube). It consists of numeric facts called measures that are categorized by dimensions. The measures are placed at the intersections of the hypercube, which is spanned by the dimensions as a vector space. The usual interface to manipulate an OLAP cube is a matrix interface, like Pivot tables in a spreadsheet program, which performs projection operations along the dimensions, such as aggregation or averaging. The cube metadata is typically created from a star schema or snowflake schema or fact constellation of tables in a relational database. Measures are derived from the records in the fact table and dimensions are derived from the dimension tables. Each measure can be thought of as

having a set of labels, or meta-data associated with it. A dimension is what describes these labels; it provides information about the measure. A simple example would be a cube that contains a store's sales as a measure, and Date/Time as a dimension. Each Sale has a Date/Time label that describes more about that sale.

For example:

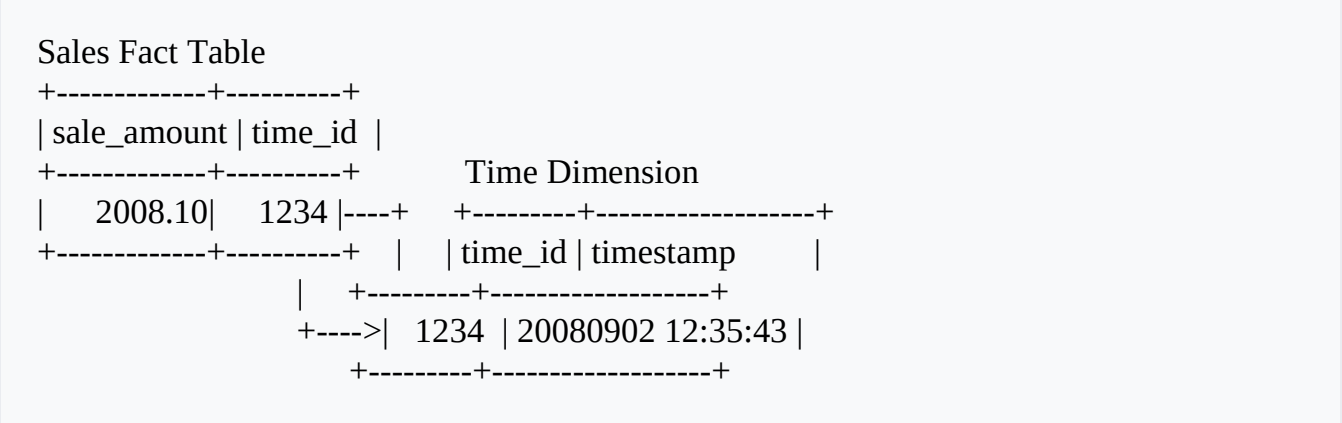


Figure 4.15: OLAP Cube

1. **Multidimensional Databases:**
2. **Multidimensional structure** is defined as a variation of the relational model that uses multidimensional structures to organize data and express the relationships between data. The structure is broken into cubes and the cubes are able to store and access data within the confines of each cube. Each cell within a multidimensional structure contains aggregated data related to elements along each of its dimensions. Even when data is manipulated it remains easy to access and continues to constitute a compact database format. The data still remains interrelated. Multidimensional structure is quite popular for analytical databases that use online analytical processing (OLAP) applications. Analytical databases use these databases because of their ability to deliver answers to complex business queries swiftly. Data can be viewed from different angles, which gives a broader perspective of a problem unlike other models.

3. Multidimensional OLAP (MOLAP)

4. MOLAP (multi-dimensional online analytical processing) is the classic form of OLAP and is sometimes referred to as just OLAP. MOLAP stores this data in an optimized multi-dimensional array storage, rather than in a relational database.
5. Some MOLAP tools require the pre-computation and storage of derived data, such as consolidations – the operation known as processing. Such MOLAP tools generally utilize a pre-calculated data set referred to as a data cube. The data cube contains all the possible answers to a given range of questions. As a result, they have a very fast response to queries. On the other hand, updating can take a long time depending on the degree of pre-computation. Pre-computation can also lead to what is known as data explosion.
6. Other **MOLAP** tools, particularly those that implement the functional database model do not pre-compute derived data but make all calculations on demand other than those that were previously requested and stored in a cache.

Advantages of MOLAP

7. Fast query performance due to optimized storage, multidimensional indexing and caching.
8. Smaller on-disk size of data compared to data stored in relational database due to compression techniques.
9. Automated computation of higher level aggregates of the data.
10. It is very compact for low dimension data sets.
11. Array models provide natural indexing.
12. Effective data extraction achieved through the pre-structuring of aggregated data.

Disadvantages of MOLAP

1. Within some MOLAP systems the processing step (data load) can be quite lengthy, especially on large data volumes. This is usually remedied by doing only

incremental processing, i.e., processing only the data which have changed (usually new data) instead of reprocessing the entire data set.

2. Some MOLAP methodologies introduce data redundancy.

4.5.5 Relational OLAP (ROLAP)

ROLAP works directly with relational databases and does not require pre-computation. The base data and the dimension tables are stored as relational tables and new tables are created to hold the aggregated information. It depends on a specialized schema design. This methodology relies on manipulating the data stored in the relational database to give the appearance of traditional OLAP's slicing and dicing functionality. In essence, each action of slicing and dicing is equivalent to adding a "WHERE" clause in the SQL statement. ROLAP tools do not use pre-calculated data cubes but instead pose the query to the standard relational database and its tables in order to bring back the data required to answer the question. ROLAP tools feature the ability to ask any question because the methodology is not limited to the contents of a cube. ROLAP also has the ability to drill down to the lowest level of detail in the database.

While ROLAP uses a relational database source, generally the database must be carefully designed for ROLAP use. A database which was designed for OLTP will not function well as a ROLAP database. Therefore, ROLAP still involves creating an additional copy of the data. However, since it is a database, a variety of technologies can be used to populate the database.

Advantages of ROLAP

1. **ROLAP** is considered to be more scalable in handling large data volumes, especially models with dimensions with very high cardinality (i.e., millions of members).

2. With a variety of data loading tools available, and the ability to fine-tune the extract, transform, load (ETL) code to the particular data model, load times are generally much shorter than with the automated MOLAP loads.
3. The data are stored in a standard relational database and can be accessed by any SQL reporting tool (the tool does not have to be an OLAP tool).
4. ROLAP tools are better at handling non-aggregatable facts (e.g., textual descriptions). MOLAP tools tend to suffer from slow performance when querying these elements.
5. By decoupling the data storage from the multi-dimensional model, it is possible to successfully model data that would not otherwise fit into a strict dimensional model.
6. The ROLAP approach can leverage database authorization controls such as row-level security, whereby the query results are filtered depending on preset criteria applied, for example, to a given user or group of users (SQL WHERE clause).

Disadvantages of ROLAP

1. There is a consensus in the industry that ROLAP tools have slower performance than MOLAP tools. However, see the discussion below about ROLAP performance.
2. The loading of aggregate tables must be managed by custom ETL code. The ROLAP tools do not help with this task. This means additional development time and more code to support.
3. When the step of creating aggregate tables is skipped, the query performance then suffers because the larger detailed tables must be queried. This can be partially remedied by adding additional aggregate tables, however it is still not practical to create aggregate tables for all combinations of dimensions/attributes.
4. ROLAP relies on the general purpose database for querying and caching, and therefore several special techniques employed by MOLAP tools are not available (such as special hierarchical indexing). However, modern ROLAP tools take advantage of latest improvements in SQL language such as CUBE and ROLLUP

operators, DB2 Cube Views, as well as other SQL OLAP extensions. These SQL improvements can mitigate the benefits of the MOLAP tools.

5. Since ROLAP tools rely on SQL for all of the computations, they are not suitable when the model is heavy on calculations which don't translate well into SQL. Examples of such models include budgeting, allocations, financial reporting and other scenarios.

4.5.6 Hybrid OLAP (HOLAP)

The undesirable trade-off between additional ETL (Extract, Transform and Load) cost and slow query performance has ensured that most commercial OLAP tools now use a Hybrid OLAP (HOLAP) approach, which allows the model designer to decide which portion of the data will be stored in MOLAP and which portion in ROLAP. There is no clear agreement across the industry as to what constitutes "Hybrid OLAP", except that a database will divide data between relational and specialized storage. For example, for some vendors, a HOLAP database will use relational tables to hold the larger quantities of detailed data, and use specialized storage for at least some aspects of the smaller quantities of more-aggregate or less-detailed data. HOLAP addresses the shortcomings of MOLAP and ROLAP by combining the capabilities of both approaches. HOLAP tools can utilize both pre-calculated cubes and relational data sources.

Comparing the different types of OLAP

1. Each type has certain benefits, although there is disagreement about the specifics of the benefits between providers.
2. Some MOLAP implementations are prone to database explosion, a phenomenon causing vast amounts of storage space to be used by MOLAP databases when certain common conditions are met: high number of dimensions, pre-calculated results and sparse multidimensional data.

3. MOLAP generally delivers better performance due to specialized indexing and storage optimizations. MOLAP also needs less storage space compared to ROLAP because the specialized storage typically includes compression techniques.
4. ROLAP is generally more scalable. However, large volume pre-processing is difficult to implement efficiently so it is frequently skipped. ROLAP query performance can therefore suffer tremendously.
5. Since ROLAP relies more on the database to perform calculations, it has more limitations in the specialized functions it can use.
6. HOLAP attempts to mix the best of ROLAP and MOLAP. It can generally pre-process swiftly, scale well, and offer good function support.

4.5.7 Other types of OLAP

The following acronyms are also sometimes used, although they are not as widespread as the earlier mentioned ones:

1. WOLAP – Web-based OLAP
2. DOLAP – Desktop OLAP
3. RTOLAP – Real-Time OLAP
4. GOLAP – Graph OLAP
5. CaseOLAP – Context-aware Semantic OLAP, developed for biomedical applications. The CaseOLAP platform includes data preprocessing (e.g., downloading, extraction, and parsing text documents), indexing and searching with Elasticsearch, creating a functional document structure called Text-Cube, and quantifying user-defined phrase-category relationships using the core CaseOLAP algorithm.

1. Conclusion:

The types of data stored in the databases were relatively simple. In the past few years, however, there has been an increasing need for handling new data types in databases, such as temporal data, spatial data, multimedia data, and geographic data and so on. Another major trend in the last decade has created its own issues, for example, the growth of mobile computers, starting with laptop computers, palmtop computers and pocket organizers. In more recent years, mobile phones have also come with built-in computers. These trends have resulted into the development of new database technologies to handle new data types and applications. In this unit, some of the emerging database technologies have been briefly introduced. We have discussed how databases are used and accessed from Internet, using web technologies, use of mobile databases to allow users widespread and flexible access to data while being mobile and multimedia databases providing support for storage and processing of multimedia information. We have also introduced how to deal with geographic information data or spatial data and their applications. Databases configured for OLAP use a multidimensional data model, allowing for complex analytical and ad hoc queries with a rapid execution time. They borrow aspects of navigational databases, hierarchical databases and relational databases. Most real OLAP databases allow you to slice data into an infinite number of dimensions - e.g. by time, product line, and sales groups. These databases are fed most often by relational databases. Many OLAP databases have their own dialect of SQL specifically designed to deal with the multidimensionality of OLAP data.

2. Tutor Marked Assignment

2. Describe the advantages and disadvantages of ROLAP
3. Describe the features of Fuzzy Databases
4. State the limitations of Classical SQL
5. Enumerate the variants of OLAP

4.5.10 Further Readings/References

Ailamaki, A., DeWitt, D., Hill, M., Wood, D.A.: DBMSs on a Modern Processor: Where Does Time Go? In Proc. 25th Int. Conf. on Very Large Databases, Edinburgh, Scotland (1999).

Bach Pedersen, Torben; S. Jensen, Christian (December 2001). "Multidimensional Database Technology". Distributed Systems Online: 40–46. ISSN 0018-9162

Borne K, Becla J, Davidson I, Szalay A, and Tyson J, 2008, "The LSST Data Mining Research Agenda," AIP Conf. Proc. 1082, pp 347-351.

Cranshaw J, Cuhadar-Donszelmann T, Gallas E, Hrivnac J, Kenyon M, McGlone H, Malon D, Mambelli M, Nowak M, Viegas F, Vinek E, and Zhang Q, 2010, "Event selection services in ATLAS," J. Phys. Conf. Ser. 219 042007.

Cranshaw J, Goosens L, Malon D, McGlone H, and Viegas F, 2008, "Building a scalable event-level metadata service for ATLAS," J. Phys. Conf. Ser. 119 072012.

Cranshaw J, Malon D. Vaniachine A, Fine V, Lauret J, and Hamill P, 2010, "Petaminer:

Daniel Lemire (December 2007). "Data Warehousing and OLAP-A Research-Oriented Bibliography".

E. Gamma, R. Helm, R. Johnson, and J. Vlissidis. Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

Erik Thomsen. (1997). OLAP Solutions: Building Multidimensional Information Systems, 2nd Edition. John Wiley & Sons. ISBN 978-0-471-14931-6.

https://www.stratoscale.com/blog/dbaas/what-is-database-as-a-service/

Hudec, M.: Fuzzy SQL for statistical databases. In MSIS, Meeting on the Management of Statistical Information Systems. Luxembourg. (2008) [Online]. Available: http://www.unece.org/stats/documents/ece/ces/ge.50/2008/wp.12.e.pdf (January 2009)

Idreos, Stratos; Kersten, Martin L; Manegold, Stefan (2007). Database cracking. Proceedings of CIDR.

International Business Machines, Corp. DB2 Spatial Extender – User's Guide and Reference, Version 8.1, 2002.

International Business Machines, Corp. Informix Spatial DataBlade, Version 8.11, 2001.

ISO/DIS 19107:2002. Geographic Information - Spatial Schema, 2002.

ISO/DIS 19111:2002. Geographic Information - Spatial Referencing by Coordinates, 2002.

ISO/IEC 9075-2:1999. Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation), 1999.

ISO/IEC 9075-2:2001 WD. Information Technology – Database Languages – SQL – Part 7: Temporal (SQL/Foundation), 2001.

ISO/IEC 9075-9:2000. Information Technology – Database Languages – SQL – Part 9: SQL/MED, 2000.

Ling Liu and Tamer M. Özsu (Eds.) (2009). "Encyclopedia of Database Systems, 4100 p. 60 illus. ISBN 978-0-387-49616-0.

Malon D, Cranshaw J, and Karr K, 2006, "A flexible, distributed event-level metadata system for ATLAS," Computing in High Energy and Nuclear Physics, Mumbai, India.

Malon D, Van Gemmeren P, Nowak M, and Schaffer A, 2006, "Schema evolution and the ATLAS event store," Computing in High Energy and Nuclear Physics, Mumbai, India.

Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications (PDF). Ph.D. Thesis. Universiteit van Amsterdam. May 2002.

Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications (PDF). Ph.D. Thesis. Universiteit van Amsterdam. May 2002.

Oracle (2004). Oracle Corporation. *www.oracle.com* OWL (2004). Web Ontology Language. *www.w3.org/ 2001/sw/WebOnt/*

P. A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, J. Teubner. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, IL, USA, June 2006.

P. A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, J. Teubner. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, IL, USA, June 2006.

Per Svensson, Peter Boncz, Milena Ivanova, Martin Kersten, Niels Nes, Doron Rotem (2010). Scientific Data Management: *Challenges, Technology and Deployment.*

Scalzo, Bert; Hotka, Dan (February 2003). Toad Handbook. Developer's Library. Jim McDaniel. Sams Publishing. p. xiv. ISBN 978-0-672-32486-4.

Siler, W., Buckley, J.: Fuzzy expert systems and fuzzy reasoning. John Wiley & Sons, Inc., New Jersey, USA. (2005)

Urrutia, A., Pavesi, L.: Extending the capabilities of database queries using fuzzy logic. In Proceedings of the Collecter-LatAm, Santiago, Chile (2004). [Online]. Available: http://www.collecter.org/archives/2004_October/06.pdf (current January 2009)

Using ROOT for efficient data storage in MYSQL database," J. Phys. Conf. Ser. 219 042036.

Van Gemmeren P and Malon D, 2009, "The event data store and I/O framework for the ATLAS experiment at the Large Hadron Collider," IEEE Int. Conf. on Cluster Computing and Workshops, p. 1.

Van Gemmeren P and Malon D, 2010, "Event metadata records as a testbed for scalable data mining", J. Phys. Conf. Ser. 219 042057.

Wang, C., Lo, A., Alhajj, R., & Barker, K. (2004). Converting legacy relational database into XML database through reserve engineering. *Proceedings of the International Conference on Enterprise Information Systems*, Portugal

Williams, C., Garza, V.R., Tucker, S, Marcus, A.M. (1994, January 24). Multidimensional models boost viewing options. InfoWorld, 16(4)

www.scidb.org/about/history.php

Zadeh, L.: Fuzzy Sets. Information and Control, No. 8, 338-353. (1965)

Zhang, Y.; Scheers, L. H. A.; Kersten, M. L.; Ivanova, M.; Nes, N. J. (2011). "Astronomical Data Processing Using SciQL, an SQL Based Query Language for Array Data". Astronomical Data Analysis Software and Systems.

Zhang, Y.; Scheers, L. H. A.; Kersten, M. L.; Ivanova, M.; Nes, N. J. (2011). "Astronomical Data Processing Using SciQL, an SQL Based Query Language for Array Data". Astronomical Data Analysis Software and Systems.