NATIONAL OPEN UNIVERSITY OF NIGERIA

FACULTY OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

COURSE CODE: CIT427

COURSE TITLE:
DATABASE SYSTEM AND MANAGEMENT

COURSE GUIDE

CIT427
DATABASE SYSTEM AND MANAGEMENT

Course Team Vivian Nwaocha (Developer/Writer) - NOUN
Prof. Hyacinth C. Inyiama (Editor) - NAU, AWKA
Prof. Kehinde Obidairo (Programme Leader) - NOUN

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Office
No. 5 Dar es Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja
Nigeria

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

Published By:
National Open University of Nigeria

First Printed 2012

Reviewed and Reprinted 2021

ISBN: 978-058-862-0

CONTENTS **PAGE**

Introduction

CIT427: Database Systems and Management is a three-credit preliminary course, intended for undergraduates studying towards acquiring the Bachelor of Science in Computer Science and other related disciplines of the National Open University of Nigeria (NOUN).

The course is divided into 5 modules and 21 study units. It offers students an introduction to the design and programming of database systems. In particular, we will cover the ER (entity-relationship) approach to data modelling, the relational model of database management systems (DBMSs) and the use of query languages such as SQL. We will also cover relational algebra and the use of SQL in a programming environment. We will briefly touch upon query processing and aspects of computer data storage and file structure. We will also dedicate the concluding module to two contemporary topics: XML Documents and Web Services.

At the end of this course, it is expected that students should be able to understand, explain and be adequately equipped to handle database systems, common storage technologies as well as XML documents and common Web Services.

The course guide therefore gives you an overview of what the course: **CIT427** is all about, the textbooks and other course materials to be referenced, what you are expected to know in each unit, and how to work through the course material. It suggests the general strategy to be adopted and also emphasises the need for self-assessment and tutor-marked assignment. There are also tutorial classes that are linked to this course and students are advised to attend.

What You Will Learn in This Course

The overall aim of this course, **CIT427**, is to enhance the students' skills in order to manage database systems effectively as well as deal with other issues related to XML and Web applications. This course provides extensive hands-on, case study examples, and reference materials designed to enhance database systems management skills. In the course of your study, you will be taught definitions of common terms, characteristics and applications of database systems. You will also learn about computer data storage and file structure. Finally, you will learn about basic concepts of XML and Web services.

Course Aim

This course aims to give students an in-depth understanding of database systems. It is hoped that the knowledge acquired from this course would enhance students' skills in handling database systems, files, common storage technologies as well as XML documents and common Web Services.

Course Objectives

It is pertinent to note that each unit has precise objectives. Students should learn them carefully before proceeding to subsequent units. Therefore, it may be useful to refer to these objectives in the course of your study of the unit to assess your progress. You should always look at the unit objectives after completing each unit. In this way, you can be sure that you have done what is required of you by the end of the unit.

However, below are overall objectives of this course. On successful completion of this course, you should be able to:

- define the term 'database management system'
- state typical examples of database management systems
- identify the categories of database management systems
- explain the concept 'database servers'
- outline the evolution of database management systems
- state the common features of database management systems
- explain the notion ' data models'
- identify the common categories of data models
- explain the concept of entity-relationship models
- describe the concept of mapping cardinalities with respect to entity-relationship diagram
- identify the link between instances and schemes
- list and describe the components of a data structure
- state the data structures required for physical system
- describe the notion of relationship sets
- mention a formal definition of a relational algebra
- state the key role of each component of an SQL expression
- identify the common forms of database modification
- discuss the link between main Constraints and Integrity Constraints
- itemise the core characteristics of storage technologies
- sketch a succinct description of computer data storage
- classify the levels of storage
- distinguish between volatile and non-volatile memory

- state the difference between the Read/Write and Read only storage
- identify the components of file organisation
- mention the key considerations in specifying a system of file organisation
- outline the components of a Structured Document
- distinguish between XML, HTML and SGML
- describe the four main kinds of declarations in XML
- list and describe the common attribute types
- identify the procedure involved in accessing information from Web Services
- mention the common advantages and disadvantages of Web Services
- list the components of Web Services Architecture.

Working through This Course

To complete this course, you are required to study all the units, the recommended text books, and other relevant materials. Each unit contains some self -assessment exercises and tutor- marked assignments. At some point in this course, you would be required to submit the tutor-marked assignments. There is also a final examination at the end of this course. Stated below are the components of this course and what you have to do.

Course Materials

The major components of the course are:

1. Course Guide
2. Study Units
3. Text Books
4. Assignment File
5. Presentation Schedule

Study Units

There are 21 study units and 5 modules in this course. They are:

Module 1 Introduction to Database Management Systems

Unit 1 Basic Concepts of Database Management Systems
Unit 2 Data Models
Unit 3 Instances and Schemes
Unit 4 Overall System Structure

Textbooks and References

These texts will be of enormous benefit to you in this course:

Avi, S.; *et al.* (N.D.). *Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (2009). (Eds.). Learning SQL (2nd Ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Borja, S. (2005). *The Globus Toolkit 4 Programmer's Tutorial.*

Byers, F. R. (2003). Care and Handling of CDs and DVDs - A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. In: *Communications of the ACM* 13 (6): 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. In: *Communications of the ACM archive*. Vol 13. Issue 6(June 1970). pp.377-387.

Database Design Basics. (N.D.).Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

“Development of an Object-Oriented DBMS”; Portland, Oregon, United States; Pages: 472 - 482; 1986; ISBN 0-89791-204-7.

Doll, S. (2002). “Is SQL a Standard Anymore?”. TechRepublic’s Builder.com. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Kawash, J. (2004). Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus - *Journal of Computers in Mathematics and Science Teaching*. 23 (2) 2004 AACE Norfolk, Virginia.

Mike, C. “Referential Integrity”. http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage and More*. Morgan Kaufmann Press,

Norman, W. (1998). *A Technical Introduction to XML.*

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBMS; Pages 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). *Database Management Systems* (2nd Ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). Beyond Relational Databases. Communications of the ACM, 51(7), 52-58. Retrieved July 6, 2009, from Business Source Complete Database.

Teorey, T.; *et al*. (2005) *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it all* (1st Ed.) Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).Database Systems: *A Practical Approach to Design, Implementation and Management*.

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

Assignment File

The assignment file will be given to you in due course. In this file, you will find all the details of the work you must submit to your tutor for marking. The marks you obtain for these assignments will count towards the final mark for the course. Altogether, there are 21 tutor-marked assignments for this course.

Presentation Schedule

The presentation schedule included in this course guide provides you with important dates for completion of each tutor-marked assignment. You should therefore endeavour to meet the deadlines.

Assessment

There are two aspects to the assessment for this course. First, there are tutor-marked assignments; and second, the written examination.

Therefore, you are expected to take note of the facts, information and problem solving gathered during the course. The tutor-marked assignments must be submitted to your tutor for formal assessment, in accordance with the deadline. The work submitted will count for 40% of your total course marks.

At the end of the course, you will need to sit for a final written examination. This examination will account for 60% of your total score.

Tutor-Marked Assignments (TMAs)

There are 21 TMAs in this course. You need to submit all the TMAs. The best four will be counted. When you have completed each assignment, send them to your tutor as soon as possible and make certain that it gets to your tutor on or before the stipulated deadline. If for any reason you cannot complete your assignment on time, contact your tutor before the assignment is due to discuss the possibility of extension. Extension will not be granted after the deadline, unless on exceptional circumstances.

Final Examination and Grading

The final examination for CIT427 will be of last for a period of 3 hours and have a value of 60% of the total course grade. The examination will consist of questions which reflect the self assessment exercise and tutor-marked assignments that you have previously encountered. Furthermore, all areas of the course will be examined. It would be better to use the time between finishing the last unit and sitting for the examination, to revise the entire course. You might find it useful to review your TMAs and comment on them before the examination. The final examination covers information from all parts of the course.

Course Marking Scheme

The following table indicates the course marking scheme:

Table 1: Course Marking Scheme

Assessment	**Marks**
Assignments 1-21	21 assignments, 40% for the best 4 Total = 10% X 4 = 40%
Final Examination	60% of overall course marks
Total	100% of Course Marks

Course Overview

This table indicates the units, the number of weeks required to complete them and the assignments.

Table 2: Course Organiser

Unit	Title of Work	Weeks Activity	Assessment (End of Unit)
	Course Guide	Week 1	
	Module 1 Introduction to Database Management Systems		
1	Basic Concepts of Database Management Systems	Week 1	Assignment 1
2	Data Models	Week 2	Assignment 2
3	Instances and Schemes	Week 3	Assignment 3
4	Overall System Structure	Week 3	Assignment 4
	Module 2 The Entity-Relationship Data Model		
1	Entities and Entity Sets	Week 4	Assignment 5
2	Relationships and Relationship Sets	Week 4	Assignment 6
3	Structure of Relational Database	Week 5	Assignment 7
4	The Relational Algebra	Week 5	Assignment 8
	Module 3 SQL and Integrity Constraints		
1	Structured Query Language (SQL) Fundamentals	Week 6	Assignment 9
2	SQL Expression	Week 6	Assignment 10
3	Database Modification	Week 7	Assignment 11
4	Integrity Constraints	Week 7	Assignment 12
	Module 4 Computer Data Storage and File Structure		
1	Computer Data Storage and Levels	Week 8	Assignment 13
2	Features of Storage Technologies	Week 9	Assignment 14
3	Common Storage Technologies	Week 10	Assignment 15
4	File Organisation		Assignment 16
	Module 5 Introduction to XML and Web Services		
1	Fundamentals of XML	Week 11	Assignment 17
2	Significance of XML	Week 12	Assignment 18
3	XML Document	Week 13	Assignment 19
4	Document Type Declaration	Week 14	Assignment 20
5	Introduction to Web Services	Week 14	Assignment 21

How to Get the Most from This Course

In open and distance learning (ODL), the study units replace the university lecturer. This is one of the huge advantages of distance learning mode; you can read and work through specially designed study materials at your own pace and at a time and place that is most convenient. Think of it as reading from the teacher, the study guide indicates what you ought to study, how to study it and the relevant texts to consult. You are provided with exercises at appropriate points, just as a lecturer might give you an in-class exercise.

Each of the study units follows a common format. The first item is an introduction to the subject matter of the unit and how a particular unit is integrated with the other units and the course as a whole. Next to this is a set of learning objectives. These learning objectives are meant to guide you in your study. The moment a unit is finished, you must go back and check whether you have achieved the objectives. If this is made a habit, then you will increase your chances of passing the course. The main body of the units also guides you through the required readings from other sources. This will usually be either from a set book or from other sources.

Self assessment exercises are provided throughout the unit, to aid personal studies and answers are provided at the end of the unit. Working through these self tests will help you to achieve the objectives of the unit and also prepare you for tutor-marked assignments and examinations. You should attempt each self test as you encounter them in the units.

The following are practical strategies for working through this course

- Read the course guide thoroughly
- Organise a study schedule. Refer to the course overview for more details. Note the time you are expected to spend on each unit and how the assignment relates to the units. Important details, e.g. details of your tutorials and the date of the first day of the semester are available. You need to gather together all these information in one place such as a diary, a wall chart calendar or an organiser. Whatever method you choose, you should decide on and write in your own dates for working on each unit.
- Once you have created your own study schedule, do everything you can to stick to it. The major reason that students fail is that they get behind with their course work. If you get into difficulties with your schedule, please let your tutor know before it is too late for help.

- Turn to Unit 1 and read the introduction and the objectives for the unit
- Assemble the study materials. Information about what you need for a unit is given in the table of content at the beginning of each unit. You will need both the study unit you are working on and one of the materials recommended for further reading, on your desk at the same time
- Work through the unit, the content of the unit itself has been arranged to provide a sequence for you to follow. As you work through the unit, you will be encouraged to read from your set books
- Keep in mind that you will learn a lot by doing all your assignments carefully. They have been designed to help you meet the objectives of the course and will help you pass the examination.
- Review the objectives of each study unit to confirm that you have achieved them. If you are not certain about any of the objectives, review the study material and consult your tutor
- When you are confident that you have achieved a unit's objectives, you can start on the next unit. Proceed unit by unit through the course and try to pace your study so that you can keep yourself on schedule
- When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor-marked assignment form and also written on the assignment. Consult your tutor as soon as possible if you have any questions or problems
- After completing the last unit, review the course and prepare yourself for the final examination. Check that you have achieved the unit objectives (listed at the beginning of each unit) and the course objectives (listed in this course guide)

Facilitation/Tutors and Tutorials

There are 8 hours of tutorial provided in support of this course. You will be notified of the dates, time and location together with the name and phone number of your tutor as soon as you are allocated a tutorial group.

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

Do not hesitate to contact your tutor by telephone, e-mail or discussion board if you need help. The following might be circumstances in which you would find help necessary: contact your tutor if:

- You do not understand any part of the study units or the assigned readings.
- You have difficulty with the self test or exercise.
- You have questions or problems with an assignment, with your tutor's comments on an assignment or with the grading of an assignment.

You should try your best to attend the tutorials. This is the only chance to have face-to- face contact with your tutor and ask questions which are answered instantly. You can raise any problem encountered in the course of your study. To gain the maximum benefit from the course tutorials, prepare a question list before attending them. You will learn a lot from actively participating in discussions.

I wish you the best in your studies!

Course Code CIT427
Course Title Database Systems and Management

Course Team Vivian Nwaocha (Developer/Writer) - NOUN
Prof. Hyacinth C. Inyiama (Editor) - NAU, AWKA
Prof. Kehinde Obidairo (Programme Leader) - NOUN

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Office
No. 5 Dar es Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja
Nigeria

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

Published By:
National Open University of Nigeria

First Printed 2012

ISBN: 978-058-862-0

CONTENTS PAGE

MODULE 1 INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

UNIT 1 BASIC CONCEPTS OF DATABASE MANAGEMENT SYSTEMS

CONTENTS

1.0 INTRODUCTION

This course provides a fundamental overview of the concepts, principles and techniques of modern database management systems and of database (data-driven) business application system development.

The most important thing you would need to grasp from this module is the fact that database management systems make the logical presentation of database information to users possible. It is much more than just learning new functions, syntax, etc. Thus, database management systems require *a logical way of thinking.*

This module has thus been designed to enhance your database management expertise. Even if you have gained previous database management experience, it is recommended that you go through the entire module systematically to gain some insight into the course.

2.0 OBJECTIVES

At the end this unit, you should be able to:

- define the term 'database management system'
- cite typical examples of database management systems
- identify the categories of database management systems
- explain the concept of 'database servers'
- outline the evolution of database management systems
- mention the common features of database management systems
- describe the main components of a database management system.

3.0 MAIN CONTENT

3.1 Overview of a Database Management System

A Database Management System (DBMS) is a set of software programs that controls the organisation, storage, management, and retrieval of data in a database. The DBMS accepts requests for data from an application program and instructs the operating system to transfer the appropriate data. The queries and responses must be submitted and received according to a format that conforms to one or more applicable protocols. When a DBMS is used, information systems can be changed much more easily as the organisation's information requirements change. New categories of data can be added to the database without disruption to the existing system.

3.1.1 What is a Database Management System?

A Database Management System can simply be defined as a set of software programs that controls the organisation, storage, management, and retrieval of data in a database.

Typical examples of Database Management Systems include Oracle Database, Microsoft SQL Server, and PostgreSQL. Nowadays, a small

number of DBMSs are used by the great majority of database applications to manage practically all the world's databases.

3.1.2 Categories of Database Management Systems (DBMS)

Database management systems are categorised according to their data structures and types. Well known types are relational, hierarchical, and object-oriented.

3.1.3 Relational Database Management System (RDBMS)

The Relational database management system organises data in tabular files. Most modern Database Management Systems (Oracle, Sybase, and Microsoft SQL Server) are relational databases. These databases support a standard language - SQL (Structured Query Language).

3.1.4 Hierarchical Database Management System (RDBMS)

This category of database management system stores data in a tree-like structure.

3.1.5 Object-Oriented Database Management System (RDBMS)

The Object-Oriented database management system stores objects as opposed to tuples or records in a RDBMS.
DBMSs are categorized according to their data structures or types.

3.1.6 Features of Database Management Systems (DBMS)

Features commonly offered by database management systems include:

Query ability

- Querying is the process of requesting attribute information from various perspectives and combinations of factors. Example: "How many 2-door cars in Texas are green?" A database query language and report writer allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data.

Backup and replication

- Copies of attributes need to be made regularly in case primary disks or other equipment fails. A periodic copy of attributes may also be created for a distant organisation that cannot readily

access the original. DBMS usually provide utilities to facilitate the process of extracting and disseminating attribute sets. When data is replicated between database servers, so that the information remains consistent throughout the database system and users cannot tell or even know which server in the DBMS they are using, the system is said to exhibit replication transparency.

Rule enforcement

- Often one wants to apply rules to attributes so that the attributes are clean and reliable. For example, we may have a rule that says each car can have only one engine associated with it (identified by Engine Number). If somebody tries to associate a second engine with a given car, we want the DBMS to deny such a request and display an error message. However, with changes in the model specification such as, in this example, hybrid gas-electric cars, rules may need to change. Ideally such rules should be able to be added and removed as needed without significant data layout redesign.

Security

- Often it is desirable to limit who can see or change which attributes or groups of attributes. This may be managed directly by individual, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groups to roles which are then granted entitlements.

Computation

- There are common computations requested on attributes such as counting, summing, averaging, sorting, grouping, cross-referencing, etc. Rather than have each computer application implement these from scratch, they can rely on the DBMS to supply such calculations.

Change and access logging

- Often one wants to know who accessed what attributes, what was changed, and when it was changed. Logging services allow this by keeping a record of access occurrences and changes.

Automated optimisation

- If there are frequently occurring usage patterns or requests, some DBMS can adjust themselves to improve the speed of those interactions. In some cases the DBMS will merely provide tools to monitor performance, allowing a human expert to make the necessary adjustments after reviewing the statistics collected.

3.1.7 Database Servers

Database servers are computers that hold the actual databases and run only the database management system and related software. They are usually multiprocessor computers, with generous memory and RAID disk arrays used for stable storage. Hardware database accelerators, connected to one or more servers via a high-speed channel, are also used in large volume transaction processing environments.

3.2 Evolution of Database Management Systems

Databases have been in use since the earliest days of electronic computing, but the vast majority of these were custom programs written to access custom databases. Unlike modern systems which can be applied to a wide range of databases and needs, these systems were tightly linked to the database in order to gain speed at the price of flexibility.

The major DBMS that evolved are as follows:

- Navigational DBMS
- Relational DBMS
- Multidimensional DBMS
- Object DBMS

3.2.1 Navigational DBMS

Objects are the basic run-time entities in an object-oriented system. A number of general-purpose database systems emerged due to growth in the speed and capability of computers; by the mid-1960s there were a number of such systems in commercial use. Interest in a standard began to grow, and Charles Bachman, founded the “Database Task Group” within CODASYL, the group responsible for the creation and standardization of COBOL. In 1971 they delivered their standard, which generally became known as the “Codasyl approach”, and soon there were a number of commercial products based on it available.

The Codasyl approach was based on the "manual" navigation of a linked data set which was formed into a large network. To find any particular record the programmer had to step through these pointers one at a time until the required record was returned. Simple queries like "find all the people in India" required the program to walk the entire data set and collect the matching results. There was, essentially, no concept of "find" or "search". This might sound like a serious limitation today, but in an era when the data was most often stored on magnetic tape such operations were too expensive to contemplate anyway.

IBM also had their own DBMS system in 1968, known as *IMS*. IMS was a development of software written for the Apollo program on the System/360. IMS was generally similar in concept to Codasyl, but used a strict hierarchy for its model of data navigation instead of Codasyl's network model. Both concepts later became known as navigational databases due to the way data was accessed, and Bachman's 1973 Turing Award presentation was *The Programmer as Navigator*. IMS is classified as a hierarchical database. IMS and IDMS, both CODASYL databases, as well as CINCOMs TOTAL database are classified as network databases.

3.2.2 Relational DBMS

Not satisfied with the navigational model of the Codasyl approach, notably the lack of a "search" facility which was becoming increasingly useful, Edgar Codd, wrote a number of papers that outlined a new approach to database construction in 1970. He described a new system for storing and working with large databases. Instead of records being stored in some sort of linked list of free-form records as in Codasyl, Codd's idea was to use a "table" of fixed-length records. A linked-list system would be very inefficient when storing "sparse" databases where some of the data for any one record could be left empty. The relational model solved this by splitting the data into a series of normalised tables, with optional elements being moved out of the main table to where they would take up room only if needed.

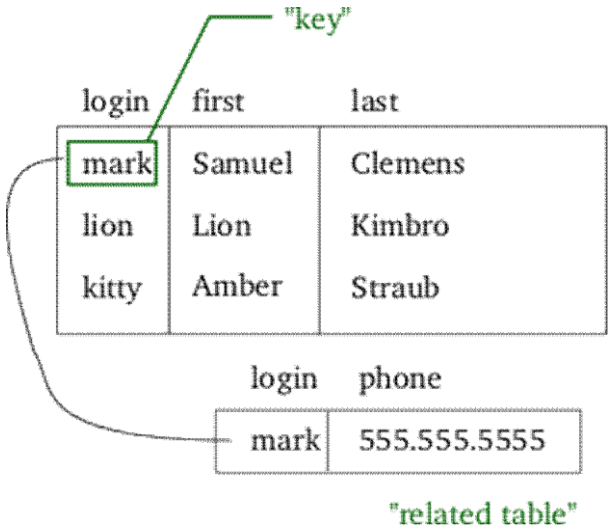

Fig. 1.1: The Relational Model

In the relational model, related records are linked together with a “key”. For instance, a common use of a database system is to track information about users, their name, login information, various addresses and phone numbers. In the navigational approach all of these data would be placed in a single record, and unused items would simply not be placed in the database. In the relational approach, the data would be *normalised* into a user table, an address table and a phone number table (for instance). Records would be created in these optional tables only if the address or phone numbers were actually provided.

Linking the information back together is the key to this system. In the relational model, some bit of information was used as a “key”, uniquely defining a particular record. When information was being collected about a user, information stored in the optional (or *related*) tables would be found by searching for this key. For instance, if the login name of a user is unique, addresses and phone numbers for that user would be recorded with the login name as its key. This “re-linking” of related data back into a single collection is something that traditional computer languages are not designed for.

Just as the navigational approach would require programs to loop in order to collect records, the relational approach would require loops to collect information about any one record. Codd’s solution to the necessary looping was a set-oriented language, a suggestion that would later spawn the ubiquitous SQL. Using a branch of mathematics known as *tuple calculus*, he demonstrated that such a system could support all the operations of normal databases (inserting, updating etc.) as well as providing a simple system for finding and returning *sets* of data in a single operation.

IBM itself did one test implementation of the relational model, PRTV, and a production one, Business System 12, both now discontinued. All other DBMS implementations usually called *relational* are actually SQL DBMSs. In 1968, the University of Michigan began development of the Micro DBMS relational database management system. It was used to manage very large data sets by the US Department of Labor, the Environmental Protection Agency and researchers from University of Alberta, the University of Michigan and Wayne State University. It ran on mainframe computers using Michigan Terminal System. The system remained in production until 1996.

3.2.3 SQL DBMS

IBM started working on a prototype system loosely based on Codd's concepts as **System R** in the early 1970s. The first version was ready in 1974/5, and work then started on multi-table systems in which the data could be split so that all of the data for a record (much of which is often optional) did not have to be stored in a single large "chunk". Subsequent multi-user versions were tested by customers in 1978 and 1979, by which time a standardized query language, SQL, had been added. Codd's ideas were establishing themselves as both workable and superior to Codasyl, pushing IBM to develop a true production version of System R, known as *SQL/DS*, and, later, *Database 2* (DB2).

Many of the people involved with INGRES became convinced of the future commercial success of such systems, and formed their own companies to commercialise the work but with an SQL interface. Sybase, Informix, Nonstop SQL and eventually Ingres itself were all being sold as offshoots to the original INGRES product in the 1980s. Even Microsoft SQL Server is actually a re-built version of Sybase, and thus, INGRES.

Stonebraker went on to apply the lessons from INGRES to develop a new database, Postgres, which is now known as PostgreSQL. PostgreSQL is often used for global mission critical applications (the .org and .info domain name registries use it as their primary data store, as do many large companies and financial institutions).

In Sweden, Codd's paper was also read and Mimer SQL was developed from the mid-70s at Uppsala University. In 1984, this project was consolidated into an independent enterprise. In the early 1980s, Mimer introduced transaction handling for high robustness in applications, an idea that was subsequently implemented on most other DBMS.

3.2.4 Object-Oriented Databases

Objects contain data, and code to manipulate that data. The entire set of The 1980s, saw a growth in how data in various databases were handled. Programmers and designers began to treat the data in their databases as objects. That is as if a person's data were in a database, that person's attributes, such as their address, phone number, and age, were now considered to belong to that person instead of being extraneous data. This allows for relationships between data to be relation to objects and their attributes and not to individual fields.

Another big game changer for databases in the 1980s was the focus on increasing reliability and access speeds. In 1989, two professors from the University of Michigan at Madison published an article at an ACM associated conference outlining their methods on increasing database performance. The idea was to replicate specific important and often queried information, and store it in a smaller temporary database that linked these key features back to the main database. This meant that a query could search the smaller database much quicker, rather than search the entire dataset. This eventually leads to the practice of indexing, which is used by almost every operating system from Windows to the system that operates Apple iPod devices.

3.2.5 Current Trends

In 1998, Researchers realised that the old trends of database management were becoming too complex and there was a need for automated configuration and management. Surajit Chaudhuri, Gerhard Weikum and Michael Stonebraker, were the pioneers that dramatically affected the thought of database management systems. They believed that database management needed a more modular approach and that there are so many specifications and needs for various users. Database management is no longer limited to "monolithic entities". Many solutions have been developed to satisfy individual needs of users. Development of numerous database options has created flexible solutions in database management.

Today there are several ways database management has affected the technology world as we know it. Organisations' demand for directory services has become an extreme necessity as they grow. Businesses are now able to use directory services that provided prompt searches for their company information. Mobile devices are not only able to store contact information of users but have grown to bigger capabilities. Mobile technology is able to cache large information that is used for computers and is able to display it on smaller devices. Web searches have even been affected with database management. Search engine

queries are able to locate data within the World Wide Web. Retailers have also benefited from the developments with data warehousing. These companies are able to record customer transactions made within their business. Online transactions have become tremendously popular with the e-business world. Consumers and businesses are able to make payments securely on company websites. None of these current developments would have been possible without the evolution of database management. Even with all the progress and current trends of database management, there will always be a need for new development as specifications and needs grow.

As the speeds of consumer internet connectivity increase, and as data availability and computing become more ubiquitous, database are seeing migration to web services. Web based languages such as XML and PHP are being used to process databases over web based services. These languages allow databases to live in "the cloud." As with many other products such as Google's Gmail, Microsoft's Office 2010, and Carbonite's online backup services, many services are beginning to move to web based services due to increasing internet reliability, data storage efficiency, and the lack of a need for dedicated IT staff to manage the hardware. Faculty at Rochester Institute of Technology published a paper regarding the use of databases in the cloud and state that their school plans to add cloud based database computing to their curriculum to "keep [their] information technology (IT) curriculum at the forefront of technology".

SELF-ASSESSMENT EXERCISE

List five common features of database management systems.

3.3 Components of Database Management System (DBMS)

A Database Management System (DBMS) includes four main parts: modeling language, data structure, database query language, and transaction mechanisms. We will look at these components in the subsequent units.

3.3.1 Modeling Language

The first component of a database management system is the implementation of a modeling language that serves to define the language of each database hosted via the DBMS. There are several approaches currently in use, with hierarchical, network, relational, and object examples. Essentially, the modeling language ensures the ability of the databases to communicate with the DBMS and thus operate on the system.

3.3.2 Data Structures

Data structures are administered by the database management system. Examples of data that are organised by this function are individual profiles or records, files, fields and their definitions, and objects such as visual media. Data structures enable DBMS to interact with the data without causing damage to the integrity of the data itself.

3.3.3 Data Query Language

A third component of DBMS software is the data query language. This element is involved in maintaining the security of the database, by monitoring the use of login data, the assignment of access rights and privileges, and the definition of the criteria that must be employed to add data to the system. The data query language works with the data structures to make sure it is harder to input irrelevant data into any of the databases in use on the system.

3.3.4 Transaction Mechanism

A database transaction mechanism ideally guarantees ACID properties in order to ensure data integrity despite concurrent user accesses (concurrency control), and faults (fault tolerance). It also maintains the integrity of the data in the database. The DBMS can maintain the integrity of the database by not allowing more than one user to update the same record at the same time. The DBMS can help prevent duplicate records via unique index constraints; for example, no two customers with the same customer numbers (key fields) can be entered into the database.

4.0 CONCLUSION

In this unit, we defined some basic concepts of database management systems. We also looked at the categories and main components of database management systems.

5.0 SUMMARY

We hope you enjoyed this unit. This unit provided an overview of database management systems: basic definition, examples, evolution, features and key components. Now, let us attempt the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. Define the term ‘database management system.’
2. List the common categories of database management systems.
3. Identify the main components of a database management system.

7.0 REFERENCES/FURTHER READING

Avi, S. *et al*. (N.D). *Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004).*Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?” *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). “Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus”. *Journal of Computers in Mathematics and Science Teaching.* Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems*(2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM,* 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009). *Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 2 DATA MODELS

CONTENTS

1.0 INTRODUCTION

This unit provides a general idea of data models as well as the different categories of data models. **Data models** are a collection of conceptual tools for describing data, data relationships, data semantics and data constraints. There are three different groups and we will look at them in more detail in the subsequent units.

2.0 OBJECTIVES

At the end this unit, you should be able to:

- explain the notion of data models
- identify common categories of data models
- describe object-based logical models
- list the common types of object-based logical models
- discuss the concept of entity-relationship models
- state the essential elements of an entity-relationship diagram
- describe the concept of mapping cardinalities with respect to entity-relationship diagram
- specify classical record-based logical models.

3.0 MAIN CONTENT

3.1 Object-Based Logical Models

Object-based logical models describe data at the conceptual and view levels. They provide fairly flexible structuring capabilities and facilitate the explicit specification of data constraints.

There are several categories of object-based logical models. They are:

- Entity-relationship model, Object-oriented model, Binary model, Semantic data model, Infological model, Functional data model, etc. However, for the purpose of this course we shall be considering the Entity-relationship and Object-oriented models.

3.1.1 The E-R Model

The entity-relationship model is based on a perception of the world as consisting of a collection of basic **objects** (entities) and **relationships** among these objects.

In order to grasp the full notion of this model, we would describe some of the key terms highlighted above. An **entity** is a distinguishable object that exists. Each entity has associated with it a set of **attributes** describing it. E.g. *number* and *balance* for an account entity. On the other hand a **relationship** is an association among several entities. For example, a *cust_acct* relationship associates a customer with each account he or she has. Thus, the set of all entities or relationships of the same type is called the **entity set** or **relationship set**.

Another essential element of the E-R diagram is the **mapping cardinalities**, which express the number of entities to which another entity can be associated with via a relationship set. We will see later how well this model works to describe real world situations.

The overall logical structure of a database can be expressed graphically by an **E-R diagram** as depicted in Figure 1.2 below:

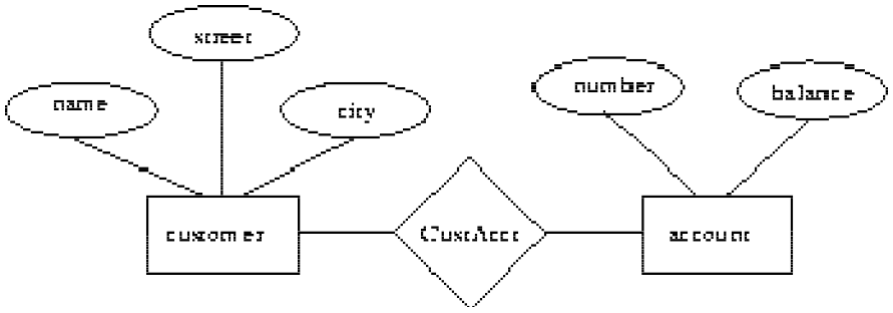

Fig. 1.2: **An E-R Diagram**

- **rectangles**: represent entity sets.
- **ellipses**: represent attributes.
- **diamonds**: represent relationships among entity sets.
- **lines**: link attributes to entity sets and entity sets to relationships.

3.1.2 The Object-Oriented Model

The object-oriented model is based on a collection of objects, like the E-R model. Classically, an object contains values stored in **instance variables** within the object. Unlike the record-oriented models, these values are themselves objects.

An object contains bodies of code that operate on the object. These bodies of code are called **methods**. Objects that contain the same types of values and the same methods are grouped into **classes**. A class may be viewed as a type definition for objects. The only way in which one object can access the data of another object is by invoking the method of that other object. This is called **sending a message** to the object. Internal parts of the object, the instance variables and method code, are not visible externally.

We can apply the aforementioned theory representing a bank account by means of an object. The object contains instance variables *number* and *balance*. The object contains a method *pay-interest* which adds interest to the balance. Under most data models, changing the interest rate entails changing code in application programs.

In the object-oriented model, this only entails a change within the *pay-interest* method. On the other hand, for entities in the E-R model, each object has its own unique identity, independent of the values it contains. Thus, two objects containing the same values are distinct. Furthermore, distinction is created and maintained in physical level by assigning distinct object identifiers.

SELF-ASSESSMENT EXERCISE

Describe the concept of mapping cardinalities with respect to entity-relationship diagram.

3.2 Record-based Logical Models

Record-based logical models are models which describe data at the conceptual and view levels. Unlike object-oriented models, they are used to specify overall logical structure of the database, and provide a higher-level description of the implementation.

They are called record-based logical models because the database is structured in fixed-format records of several types. Each record type defines a fixed number of fields, or attributes. Each field is usually of a fixed length (this simplifies the implementation). Record-based models do not include a mechanism for direct representation of code in the database. Separate languages associated with the model are used to express database queries and updates.

The three most widely accepted models are the **relational, network**, and **hierarchical**. We will now briefly consider these models in the units that follow.

3.2.1 The Relational Model

In the relational model, data and relationships are represented by a collection of **tables**. Each **table** has a number of columns with unique names, e.g. *customer, account*. Figure 1.3 shows a sample relational database.

NAME	STREET	CITY	NUMBER
Bisi	Agbowo	Ibadan	NOU090
Hassan	Ahmadu Bello	Lagos	NOU056
Onome	Atama	Benin	NOU074
Uduak	Rowlings	Calabar	NOU081
Zainab	Sule	Katsina	NOU064

NAME	BALANCE
NOU090	3000
NOU056	10000
NOU074	14000
NOU081	20000
NOU064	18000

Fig. 1.3: A Sample Relational Database

3.2.2 The Network Model

In the network model, data are represented by collections of records. Relationships among data are represented by links. A network model is typically arranged as an **arbitrary graph**. Figure 1.4 shows a sample network database

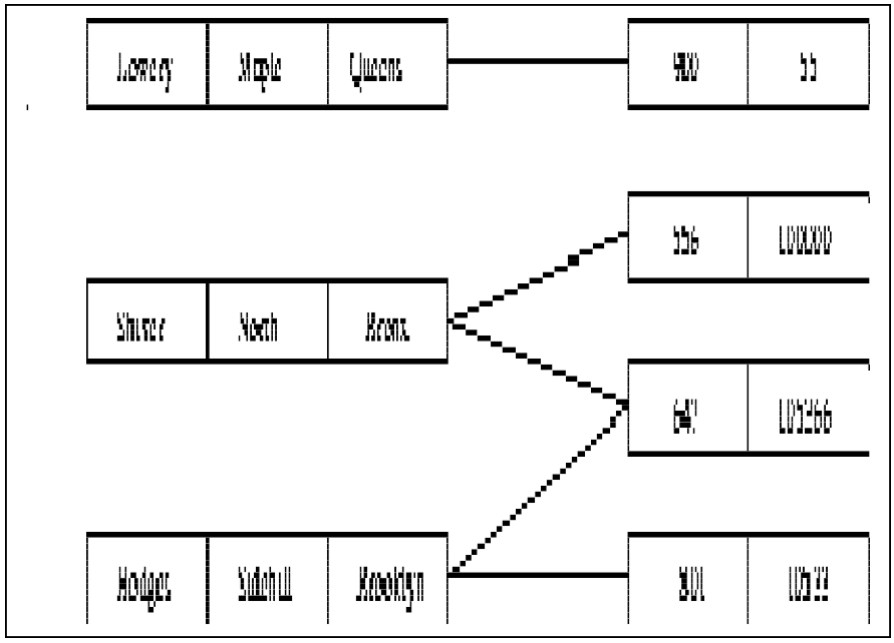

Fig. 1.4: **A Sample Network Database**

3.2.3 The Hierarchical Model

The hierarchical model is similar to the network model. However, in this model organisation of the records is as a collection of **trees**, rather than arbitrary graphs.

The relational model does not use pointers or links, but relates records by the values they contain. This allows a formal mathematical foundation to be defined. Figure 1.5 below shows a sample hierarchical database.

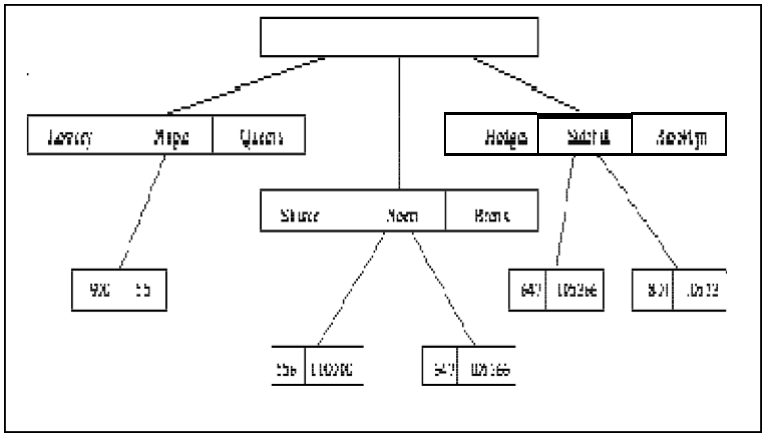

Fig. 1.5: **A Sample Hierarchical Database**

3.3 Physical Data Models

Physical data models are used to describe data at the lowest level. There are very few physical data models, e.g. Unifying model, Frame memory. It would suffice to mention these two since these models are outside the scope of our studies.

4.0 CONCLUSION

From our studies in this unit, it is vital to remember that Object-based logical models describe data at the conceptual and view levels. Two common types of object-based logical models are Entity-relationship models and Object-oriented models. It is equally worth noting that each data model plays a specific role following a specific line of action.

5.0 SUMMARY

In this unit, we introduced the concept of data models, identified the common categories of data models. We equally described the notion of entity-relationship models, stating the essential elements of an entity-relationship diagram. Furthermore, we described the concept of mapping cardinalities with respect to entity-relationship diagram. We hope you found the unit enlightening. To assess your comprehension, attempt the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. Describe object-based logical models.
2. List the common types of object-based logical models.
3. Explain the concept of entity-relationship models.
4. State the essential elements of an entity-relationship diagram.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D).*Database System Concepts*(6th ed.). McGraw-Hill.

Beaulieu, A. & Mary, E. T .(Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems*. (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?” *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S. *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More.* Morgan Kaufmann Press.

Kawash, J. (2004). “Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus”. *Journal of Computers in Mathematics and Science Teaching.* Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM,* 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 3 INSTANCES AND SCHEMES

CONTENTS

1.0 INTRODUCTION

The initial task we have in this unit is to describe the notion of instances and schemes, in this way you will gain a broader understanding of the mode of change of data over time. In sum, you would be introduced to data manipulation and other associated terms.

2.0 OBJECTIVES

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

- describe Instances and Schemes
- determine the link between instances and schemes
- identify the classical categories of schemes
- specify the duties of a Database Administrator
- list the common Database Users.

3.0 MAIN CONTENT

3.1 Concepts of Instances and Schemes

The information in a database at a particular point in time is called an **instance** of the database. While the overall design of the database is called the database **scheme**.

3.1.1 Link between Instances and Schemes

In order to grasp the key aspects of Instances and Schemes we would identify the link between these two concepts. Instances and Schemes are two terms closely associated with the mode of change database over time.

3.1.2 Analogy with Programming Languages

Instances and Schemes correlate with programming languages as follows:

- Data type definition - scheme
- Value of a variable - instance

3.1.3 Categories of Schemes

There are several classes of schemes, corresponding to the levels of **abstraction**:

- Physical scheme
- Conceptual scheme
- Sub scheme (can be many)

Having been introduced to instances and schemes, we would now consider other concepts associated with instances and schemes.

3.2 Data Independence

Data independence refers to the ability to modify a scheme definition in one level without affecting a scheme definition in a higher level.

3.2.1 Classes of Data Independence

There are two categories of data independence:

Physical data independence

- The ability to modify the physical scheme without causing application programs to be rewritten
- Modifications at this level are usually to improve performance

Logical data independence

- The ability to modify the conceptual scheme without causing application programs to be rewritten
- Usually done when logical structure of database is altered
- Logical data independence is harder to achieve as the application programs are to a large extent heavily dependent on the logical structure of the data.

3.3 Data Definition Language (DDL)

The **data definition language** (DDL) is a language used to specify a database scheme as a set of definitions expressed in a DDL. DDL statements are compiled, resulting in a set of tables stored in a special file called a **data dictionary** or **data directory**. This directory contains **metadata** (data about data). The storage structure and access methods used by the database system are specified by a set of definitions in a special type of DDL called a **data storage and definition** language. Basically, DDL statements enable developers hide the implementation details of the database schemes from the users.

3.4 Data Manipulation Language (DML)

3.4.1 Data Manipulation

Data Manipulation could refer to any of the following:

- **retrieval** of information from the database
- **insertion** of new information into the database
- **deletion** of information in the database
- **modification** of information in the database

3.4.2 Data Manipulation Language (DML)

A **Data Manipulation Language (DML)** is a language which enables users to access and manipulate data. The main goal of the DML is to provide efficient human interaction with the system.

3.4.3 Types of Data Manipulation Language (DML)

There are two types of DML:

- **procedural**: the user specifies *what* data is needed and *how* to get it
- **nonprocedural**: the user only specifies *what* data is needed

Easier for user

- May not generate code as efficient as that produced by procedural languages.
- The terms DML and query language are often used synonymously. This is due to the fact that a **query language** is a portion of a DML involving information retrieval only.

SELF-ASSESSMENT EXERCISE

Identify four classical forms of data manipulation.

3.5 Database Administrator

The term **Database Administrator** simply refers to a **person** having central control over data and programs accessing that data.

3.5.1 Duties of the Database Administrator

The duties of the database administrator include:

- **Scheme definition:** the creation of the original database scheme. This involves writing a set of definitions in a DDL (data storage and definition language), compiled by the DDL compiler into a set of tables stored in the data dictionary.
- **Storage structure and access method definition:** writing a set of definitions translated by the data storage and definition language compiler
- **Scheme and physical organisation modification:** writing a set of definitions used by the DDL compiler to generate modifications to appropriate internal system tables (e.g. data dictionary). This is done rarely, but sometimes the database scheme or physical organisation must be modified.
- **Granting of authorisation for data access:** granting different types of authorisation for data access to various users
- **Integrity constraint specification:** generating integrity constraints. These are consulted by the database manager module whenever updates occur.

3.6 Database Users

The **database users** fall into several categories:

3.6.1 Application Programmers

Application programmers are computer professionals interacting with the system through DML calls embedded in a program written in a host language (e.g. C, PL/1, Pascal). These programs are called **application programs**.

The **DML precompiler** converts DML calls (prefaced by a special character like $, #, etc.) to normal procedure calls in a host language. The host language compiler then generates the object code. Some special types of programming languages combine Pascal-like control structures with control structures for the manipulation of a database. These are sometimes called **fourth-generation languages**. They often include features to help generate forms and display data.

3.6.2 Sophisticated Users

Sophisticated users interact with the system without writing programs. They form requests by writing queries in a database query language. These are submitted to a **query processor** that breaks a DML statement down into instructions for the database manager module.

3.6.3 Specialised Users

Specialised users are sophisticated users writing special database application programs. These may be CADD systems, knowledge-based and expert systems, complex data systems (audio/video), etc.

3.6.4 Naive Users

Naive users are unsophisticated users who interact with the system by using permanent application programs (e.g. automated teller machine).

4.0 CONCLUSION

Instances and Schemes are two terms closely associated with the mode of change database over time. The information in a database at a particular point in time is called an **instance** of the database. While the overall design of the database is called the database **scheme**. A database Administrator is a **person** having central control over data and programs and programs accessing that data. The main goal of the DML is to provide efficient human interaction with the system.

5.0 SUMMARY

In this unit, we considered instances and schemes, stating the classical categories of schemes. We equally looked at data definition language and data manipulation language as well as database users and duties of a database administrator. Hoping that you understood the topics discussed, you may now attempt the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. State the categories of data independence.
2. How would a database administrator define a scheme?
3. Describe a data dictionary.
4. How do sophisticated users interact with database systems?

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D).*Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems*. (3rd Ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S.(2002). “Is SQL a Standard Anymore?” *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More.* Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching.* Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 4 OVERALL SYSTEM STRUCTURE

CONTENTS

1.0 INTRODUCTION

In the previous unit we discussed instances and schemes, data manipulation and data definition language. This unit presents the components of data structure and data structures required for physical system implementation.

2.0 OBJECTIVES

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

- explain the basis of partitioning databases
- list and describe the components of a data structure
- state the data structures required for physical system implementation.

3.0 MAIN CONTENT

3.1 Partitioning of Databases

Database systems are normally partitioned into modules for different functions. Some functions (e.g. file systems) may be provided by the operating system.

3.2 Components of Data Structure

The components of a data structure include:

- **File manager** manages allocation of disk space and data structures used to represent information on disk.
- **Database manager**: The interface between low-level data and application programs and queries.
- **Query processor** translates statements in a query language into low-level instructions the database manager understands. (May also attempt to find an equivalent but more efficient form.)
- **DML precompiler** converts DML statements embedded in an application program to normal procedure calls in a host language. The precompiler interacts with the query processor.
- **DDL compiler** converts DDL statements to a set of tables containing metadata stored in a data dictionary.

SELF-ASSESSMENT EXERCISE

What is the basis for partitioning databases?

3.3 Data Structures for Physical Implementation

Several data structures are required for physical system implementation, these include:

- **Data files:** store the database itself.
- **Data dictionary:** stores information about the structure of the database. It is used **heavily**. Great emphasis should be placed on developing a good design and efficient implementation of the dictionary.
- **Indices:** provide fast access to data items holding particular values.

Figure 1.6 below depicts these components:

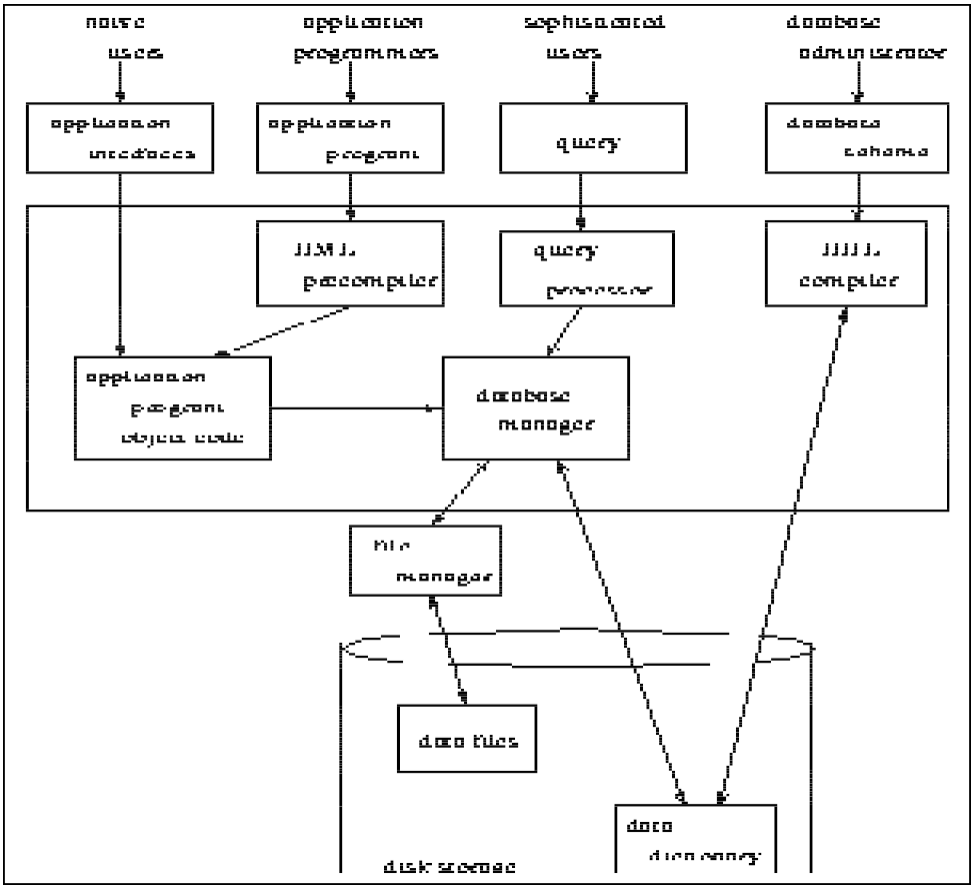

Fig. 1.6: Database System Structure

4.0 CONCLUSION

Note that, a typical data structure consists of file manager, database manager, query processor, data manipulation language precompiler and the data definition language compiler. Database systems are partitioned into modules for different functions. Data structures required for physical system implementation include data files, indices and data dictionary.

5.0 SUMMARY

This unit provided us with databases and data structures. The components a data structure was equally highlighted. In order to assess your understanding of this unit, you need to attempt the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. Describe any three components of a data structure.
1. Explain how statements in a query language are translated.
2. State the data structures required for physical system implementation.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D). *Database System Concepts*(6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004).*Database Systems* (3rd ed.).Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E.F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E.F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?”. *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching.* Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM,* 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

MODULE 2 THE ENTITY-RELATIONSHIP DATA MODEL

UNIT 1 ENTITIES AND ENTITY SETS

CONTENTS

1.0 INTRODUCTION

The **E-R** (entity-relationship) data model views the real world as a set of basic **objects** (entities) and **relationships** among these objects. It is intended primarily for the database design process by allowing the specification of an **enterprise scheme**. This represents the overall logical structure of the database. In this unit, we will consider entities, types and representation. A brief analogy would be made with respect to programming languages.

2.0 OBJECTIVES

At the end this unit, you should be able to:

- describe an entity
- specify the common types of entities
- explain the concept of an entity set
- state the attributes for entity representation
- identify the link between entities and programming languages.

3.0 MAIN CONTENT

3.1 What is an Entity?

An **entity** simply refers to an object that exists and is distinguishable from other objects. For instance, John Harris with S.I.N. 890-12-3456 is an entity, as he can be uniquely identified as one particular person in the universe.

3.2 Types of Entities

An entity may be **concrete** (a person or a book, for example) or **abstract** (like a holiday or a concept).

3.3 Entity Set

An **entity set** is a set of entities of the same type (e.g., all persons having an account in a particular bank).

Entity sets **need not be disjoint**. For example, the entity set *employee* (all employees of a bank) and the entity set *customer* (all customers of the bank) may have members in common.

SELF-ASSESSMENT EXERCISE

Specify the two types of entities.

3.4 Entity Representation

An entity is represented by a set of **attributes** as follows:

- E.g. name, S.I.N., street, city for "customer"' entity.
- The **domain** of the attribute is the set of permitted values (e.g. the telephone number must be seven positive integers).
- An attribute is in fact a **function** which maps an entity set into a domain. Consequently every entity is described by a set of (attribute, data value) pairs. Normally, there is one pair for each attribute of the entity set. For example, a particular *customer* entity is described by the set {(name, Harris), (S.I.N., 890-123-456), (street, North), (city, Georgetown)}.

3.5 Entity and Programming Language

An analogy can be made with the programming language notion of type definition. The concept of an **entity set** corresponds to the programming language **type definition**. A variable of a given type has a particular

value at a point in time. Thus, a programming language variable corresponds to an **entity** in the E-R model.

4.0 CONCLUSION

We discovered that the **E-R** (entity-relationship) data model is intended primarily for the database design process by allowing the specification of an **enterprise scheme**. This represents the overall logical structure of the database. The concept of an **entity set** corresponds to the programming language **type definition**. A variable of a given type has a particular value at a point in time. Thus, a programming language variable corresponds to an **entity** in the E-R model. An entity is represented by a set of **attributes.**

5.0 SUMMARY

In this unit, we learnt about the entity-relationship data model, giving a concise description of an entity. We equally considered entity types and representations. A brief analogy was made with respect to programming languages. Be assured that the facts gathered from this unit will be valuable for understanding other aspects of database systems. Now, attempt the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. Explain the concept of an entity set.
2. State the link between entities and programming languages.
3. Within the context of entity representation, what is an attribute?

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D).*Database System Concepts* (6th ed.). McGraw-Hill.

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems*. (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E.F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM archive*. Vol 13. Issue 6.pp. 377-387

"Database Design Basics". (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). "Is SQL a Standard Anymore?" *TechRepublic's Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching* ISSN 0731-9258 0731-9258 Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). *Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM,* 51(7),pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design.* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al.* (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 2 RELATIONSHIPS AND RELATIONSHIP SETS

CONTENTS

1.0 INTRODUCTION

In database systems, relationships and entities are very important. The notion of mapping cardinalities equally plays a significant role. This unit sheds more light on the aforementioned topics.

2.0 OBJECTIVES

At the end this unit, you should be able to:

- define a relationship
- describe the notion of relationship sets
- explain the role of an entity
- state the Entity-Relationship scheme for defining constraints
- identify primary keys for relationship sets.

3.0 MAIN CONTENT

3.1 Relationships

A **relationship** is a term which simply describes an association between several entities.

3.2 Relationship Set

A **relationship set** is a set of relationships of the same type.

In formal terms, it is a mathematical relation on $n \geq 2$ (possibly non-distinct) sets.

If $E_1, E_2, \ldots, E_n$ are entity sets, then a relationship set R is a **subset** of

$$\{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in En\}$$

Where $(e_1, e_2, \ldots, e_n)$ is a relationship.

To illustrate this further, consider two entity sets *customer* and *account*. We define the relationship *CustAcct* to denote the association between customers and their accounts. This is a **binary** relationship set.

Now, going back to our formal definition, the relationship set *CustAcct* is a subset of all the possible customer and account pairings. This is a binary relationship. Occasionally there are relationships involving more than two entity sets.

SELF-ASSESSMENT EXERCISE

How would you specify a relationship set in formal terms?

3.3 The Role of an Entity

The **role** of an entity is the function it plays in a relationship. For example, the relationship *works-for* could be ordered pairs of *employee* entities. The first employee takes the role of manager, and the second one will take the role of worker.

A relationship may also have **descriptive** attributes. For example, *date* (last date of account access) could be an attribute of the *CustAcct* relationship set.

3.4 Mapping Constraints

An Entity-Relationship (E-R) scheme may define certain constraints to which the contents of a database must conform.

Mapping Cardinalities: express the number of entities to which another entity can be associated via a relationship. For binary relationship sets between entity sets A and B, the mapping cardinality must be one of:

- **One-to-one**: An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A. (Figure 2.3)
- **One-to-many**: An entity in A is associated with any number in B. An entity in B is associated with at most one entity in A. (Figure 2.4)
- **Many-to-one**: An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A. (Figure 2.5)
- **Many-to-many**: Entities in A and B are associated with any number from each other. (Figure 2.6)

The appropriate mapping cardinality for a particular relationship set depends on the real world being modeled. (Think about the *CustAcct* relationship...)

- **Existence Dependencies:** if the existence of entity X depends on the existence of entity Y, then X is said to be **existence dependent** on Y. (Or we say that Y is the **dominant** entity and X is the **subordinate** entity.)

For example:

- Consider *account* and *transaction* entity sets, and a relationship *log* between them.
- This is one-to-many from account to transaction.
- If an *account* entity is deleted, its associated *transaction* entities must also be deleted.
- Thus *account* is dominant and *transaction* is subordinate.

3.5 Primary Keys for Relationship Sets

The attributes of a relationship set are the attributes that comprise the primary keys of the entity sets involved in the relationship set.

For example:

- *S.I.N.* is the primary key of *customer*, and
- *Account-number* is the primary key of *account*.
- The attributes of the relationship set *custacct* are then *(account-number, S.I.N.)*.

This is enough information to enable us to relate an account to a person. If the relationship has descriptive attributes, those are also included in its attribute set. For example, we might add the attribute *date* to the above relationship set, signifying the date of last access to an account by a particular customer. Note that this attribute cannot instead be placed in either entity set as it relates to both a customer and an account, and the relationship is many-to-many.

The primary key of a relationship set R depends on the **mapping cardinality** and the presence of **descriptive attributes**.
With no descriptive attributes:

- **Many-to-many:** all attributes in R.
- **One-to-many:** primary key for the "many" entity.

Descriptive attributes may be added, depending on the mapping cardinality and the semantics involved.

4.0 CONCLUSION

Relationships describe association between several entities. A relationship set is a set of relationships of the same type. In formal terms, it is a mathematical relation on $n \geq 2$ (possibly non-distinct) sets. The role of an entity is the function it plays in a relationship. Mapping Cardinalities express the number of entities to which another entity can be associated via a relationship.

5.0 SUMMARY

In summary, this unit looked at relationships and relationship sets, identifying the role of an entity and the primary keys for relationship sets. We can now attempt the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. Give a brief description of relationship sets.
2. Explain the role of an entity.
3. State the Entity-Relationship scheme for defining constraints.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D).*Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?” *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. (1st Ed.). Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). “Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus”. *Journal of Computers in Mathematics and Science Teaching* Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009). *Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 3 STRUCTURE OF RELATIONAL DATABASE

CONTENTS

1.0 INTRODUCTION

The previous unit introduced relationships and relationship sets and entities. We will consider specific entity-relationships in this unit.

2.0 OBJECTIVES

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

- define a logical structure of a relational database by means of an E-R diagram
- list the components of an E-R diagram
- explain the roles of each component of an E-R diagram.

3.0 MAIN CONTENT

3.1 The Entity Relationship Diagram

We can express the overall logical structure of a relational database graphically by means of an Entity-Relationship (E-R) diagram.

3.2 Components of an Entity Relationship Diagram

The components of an entity-relationship diagram and their corresponding roles are as follows:

- rectangles representing entity sets
- ellipses representing attributes
- diamonds representing relationship sets
- lines linking attributes to entity sets and entity sets to relationship sets.

In the course of our studies, lines would be directed (either with an arrow at the end) to signify mapping cardinalities for relationship sets. The Figures below illustrate this:

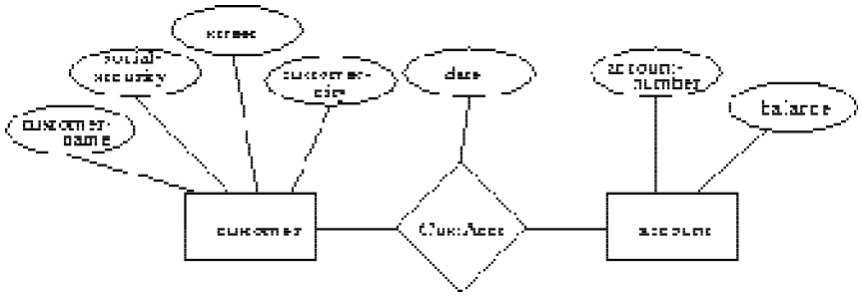

Fig. 2.1: An E-R Diagram

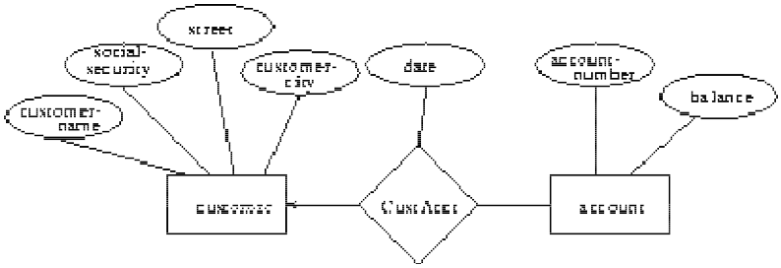

Fig. 2.2: One-To-Many from Customer to Account

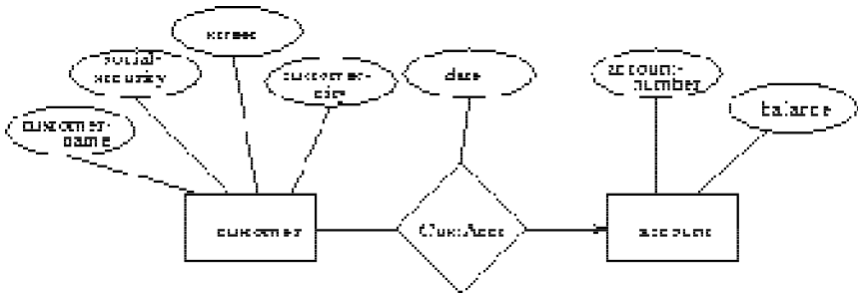

Fig. 2.3: Many-To-One from Customer to Account

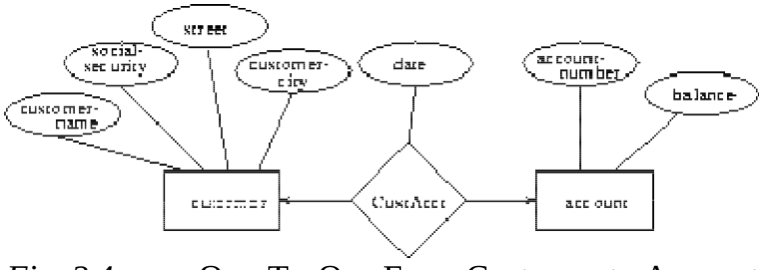

Fig. 2.4: One-To-One From Customer to Account

In order to grasp the aspect of relationship better, it is recommended that you review mapping cardinalities treated in the previous unit. They

express the number of entities to which an entity can be associated via a relationship.

The arrow positioning is simple. Simply think of the arrow head as pointing to the entity that "one" refers to.

SELF-ASSESSMENT EXERCISE

State any four components of an entity-relationship model and their corresponding key roles.

3.3 Design of an E-R Database Scheme

The E-R data model provides a wide range of choice in designing a database scheme to accurately model some real-world situation.

Some of the decisions to be made are:

- Using a ternary relationship versus two binary relationships.
- Whether an entity set or a relationship set best fit a real-world concept.
- Whether to use an attribute or an entity set.
- Use of a strong or weak entity set.
- Appropriateness of generalisation.
- Appropriateness of aggregation.

4.0 CONCLUSION

In conclusion, the entire logical structure of a relational database can be represented **graphically** by means of an **Entity-Relationship (E-R)** diagram. Components of an entity-relationship diagram and their corresponding roles are as follows:

- **rectangles** representing entity sets
- **ellipses** representing attributes
- **diamonds** representing relationship sets
- **lines** linking attributes to entity sets and entity sets to relationship sets.

5.0 SUMMARY

In this unit, we considered specific entity-relationship diagrams and their components. We equally looked at models for designing E-R database schemes. Now, please attempt the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. Outline the logical structure of a relational database by means of an E-R diagram.
2. List the components of an E-R diagram.
3. State the specific roles of each component of an E-R diagram.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D).*Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T.(Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems*. (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?” *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching.* Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM,* 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design*. (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 4 THE RELATIONAL ALGEBRA

CONTENTS

1.0 INTRODUCTION

This unit presents the relational algebra. The fundamental operations are highlighted and the outcome that corresponds to each operation is specified. As you study this unit, you need to take note of these key points.

2.0 OBJECTIVES

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

- define relational algebra
- mention six basic operations
- state the specific role of each operation
- identify the operators that correspond to each operation.

3.0 MAIN CONTENT

3.1 Formal Definition of a Relational Algebra

Basically, the **relational algebra** is a procedural query language. It can be defined formally by means of a basic expression consisting of either:

- a relation in the database
- a constant relation

General expressions of a relational algebra are formed out of smaller sub-expressions using

- $\sigma_p(E_1)$ select (p a predicate)
- $\Pi_s(E_1)$ project (s a list of attributes)
- $\rho_x(E_1)$ rename (x a relation name)
- $E_1 \cup E_2$ union
- $E_1 - E_2$ set difference
- $E_1 \times E_2$ cartesian product

SELF-ASSESSMENT EXERCISE

How are relational algebras defined?

3.2 Fundamental Operations

Relational algebra necessarily undertakes operations. These operations in turn produce a new relation as a result. There are basically six operations a relational algebra can undergo, these include:

- select (unary)
- project (unary)
- cartesian product (binary)
- rename (unary)
- union (binary)
- set-difference (binary)

Now, we shall consider these operations in-depth in the units that ensue.

3.2.1 The Select Operation

T**he Select** operation is responsible for selecting **tuples** that satisfy a given predicate. This operation is denoted by a lowercase Greek sigma (σ), with the predicate appearing as a subscript. The argument relation is given in parentheses following the σ.

For example, to select tuples (rows) of the *borrow* relation where the branch is “SFU”, we would write:

$$\sigma_{branch = \text{“SFU”}}(borrow)$$

Figure 2.5.represent the *borrow* and *branch* relations in the banking example:

bname	loan#	cname	amount
Downtown	17	Jones	1000
Longheed.Mall	23	Smith	2000
SFU	15	Hayes	1500

bname	assets	bcity
Downtown	9,000,000	Vancouver
LongheedMall	21,000,000	Burnaby
SFU	17,000,000	Burnaby

Fig. 2.5: The Borrow and Branch Relations

The new relation created as the result of this operation consists of one tuple: $(SFU, 15, Hayes, 1500)$.

We allow comparisons using =, $\neq$, <, $\leq$, > and $\geq$in the selection predicate.
We also allow the logical connectives $\vee$(or) and $\wedge$(and). For example:

$$\sigma_{bname="Downtown" \wedge amount>1200}(borrow)$$

cname	banker
Hayes	Jones
Johnson	Johnson

Fig. 2.6: The *Client* Relation

Suppose there is one more relation, *client*, shown in Figure 2.6 with the scheme.

$$Client\_scheme = (cname, banker)$$

we might write

$$\sigma_{cname=banker}(client)$$

to find clients who have the same name as their banker.

3.2.2 The Project Operation

Project copies its argument relation for the specified attributes only. Since a relation is a **set**, duplicate rows are eliminated.

Projection is denoted by the Greek capital letter pi ( ). The attributes to be copied appear as subscripts.

For example, to obtain a relation showing customers and branches, but ignoring amount and loan#, we write

$$\Pi_{bname,cname}(borrow)$$

We can perform these operations on the relations resulting from other operations. To get the names of customers having the same name as their bankers,

$$\Pi_{cname}(\sigma_{cname=banker}(client))$$

Think of **select** as taking rows of a relation, and **project** as taking columns of a relation.

3.2.3 The Cartesian Product Operation

The **cartesian product** of two relations is denoted by a cross ($\times$), written

$$r_1 \times r_2 \text{ for relations } r_1 \text{ and } r_2$$

The result of $r_1 \times r_2$ is a new relation with a tuple for each possible **pairing** of tuples from r_1 and r_2.

In order to avoid ambiguity, the attribute names have been attached to the name of the relation from which they came. If no ambiguity will result, we drop the relation name.

The result $client \times customer$ is a very large relation. If r_1 has n_1 tuples, and r_2 has n_2 tuples, then $r = r_1 \times r_2$ will have $n_1 n_2$ tuples.

The resulting scheme is the concatenation of the schemes of r_1 and r_2, with relation names added as mentioned.

To find the clients of banker Johnson and the city in which they live, we need information in both *client* and *customer* relations. We can get this by writing

$$\sigma_{banker="Johnson"}(client \times customer)$$

However, the *customer.cname* column contains customers of bankers other than Johnson. (Why?)

We want rows where *client.cname* = *customer.cname*. So we can write

$$\sigma_{client.cname=customer.cname}(\sigma_{banker="Johnson"}(client \times customer))$$

to get just these tuples.

Finally, to get just the customer's name and city, we need a projection:

$$\Pi_{client.cname,city}(\sigma_{client.cname=customer.cname}(\sigma_{banker="Johnson"}(client \times customer)))$$

3.2.4 The Rename Operation

The **rename** operation solves naming problems which occur when performing the Cartesian product of a relation with itself.

Suppose we want to find the names of all the customers who live on the same street and in the same city as Smith. We can get the street and city of Smith by writing

$$\Pi_{street,ccity}\left(\sigma_{cname="Smith"}\left(customer\right)\right)$$

To find other customers with the same information, we need to reference the *customer* relation again:

$$\sigma p\left(customer \times \left(\Pi_{street,ccity}\ \left(\sigma_{cname="Smith"}\left(customer\right)\right)\right)\right)$$

Where P is a selection predicate requiring *street* and *city* values to be equal.

3.2.5 The Union Operation

The **union** operation is denoted $\cup$ as in set theory. It returns the union (set union) of two compatible relations.

For a union operation $r \cup s$ to be legal, we require that:

- r and s must have the same number of attributes.
- The domains of the corresponding attributes must be the same.
- To find all customers of the SFU branch, we must find everyone who has a loan or an account or both at the branch. We need both *borrow* and *deposit* relations for this:

$$\Pi_{cname}\left(\sigma_{bname="SFU"}\left(borrow\right)\right) \cup \Pi_{cname}\left(\sigma_{bname="SFU"}\left(deposit\right)\right)$$

As in all set operations, duplicates are eliminated, giving the relation of Figure 3.5(a).

Fig. 3.5: The Union and Set-Difference Operations

3.2.6 The Set Difference Operation

Set difference is denoted by the minus sign (_). It finds tuples that are in one relation, but not in another.

Thus $r - s$ results in a relation containing tuples that are in r but not in s.

To find customers of the SFU branch who have an account there but no loan, we write

$$\Pi_{cname}(\sigma_{bname="SFU"}(deposit)) - \Pi_{cname}(\sigma_{bname="SFU"}(borrow))$$

The result is shown in Figure 3.5(b).
We can do more with this operation. Suppose we want to find the largest account balance in the bank.

The following strategy should be adopted:

- Find a relation r containing the balances **not** the largest.
- Compute the set difference of r and the *deposit* relation.
- To find r, we write

$$\Pi_{deposit.balance}(\sigma_{deposit.balance<d.balance}(deposit \times \rho_d(deposit)))$$

This resulting relation contains all balances except the largest one. (See Figure 3.6(a)).

Now we can finish our query by taking the set difference:

$$\Pi_{balance}(deposit) - \Pi_{deposit.balance}(\sigma_{deposit.balance<d.balance}(deposit \times \rho_d(deposit)))$$

Figure 2.7 (b) shows the result.

(a)

balance
100
300
700

(b)

balance
1300

Fig. 2.7: Find the Largest Account Balance in the Bank

4.0 CONCLUSION

To wrap up, recall that the **state** of an object is the property which describes its individual data or unique configuration. A class-scoped variable is defined by making a declaration within the class' code block but outside of any methods or properties. Properties of instantiated objects are accessed using the object name followed by the member access operator (.) and the property name.

5.0 SUMMARY

This unit provided an overview of relational algebras, stating the roles of the basic operations and the operator that corresponds to each operation. In order to assess your level of assimilation, you need to answer the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. Give a formal definition of a relational algebra.
3. List at least four basic operations.
4. State the specific role of the project operation.
5. Identify the operator that correspond to the union operation.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D). *Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems*. (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). "Is SQL a Standard Anymore?". *TechRepublic's Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching*. Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th Ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

MODULE 3 SQL AND INTEGRITY CONSTRAINTS

UNIT 1 STRUCTURED QUERY LANGUAGE (SQL) FUNDAMENTALS

CONTENTS

1.0 INTRODUCTION

This unit delves into an important aspect of a database management system- the Structured Query language (SQL). We will briefly consider the structural components of the Structured Query language.

2.0 OBJECTIVES

At the end this unit, you should be able to:

- identify the structural components of SQL
- state the specific roles of a data definition language
- describe the concept of an interactive data manipulation language.

3.0 MAIN CONTENT

3.1 Structured Query Language (SQL)

For some time now, the Structured Query Language (SQL) has been in use and has gradually evolved to become the standard relational database language.

3.2 Structural Components of SQL

A typical SQL consists of several parts which include:

Data definition language (DDL) - provides commands to

- Define relation schemes.
- Delete relations.
- Create indices.
- Modify schemes.

- **Interactive data manipulation language (DML)** - a query language based on both relational algebra and tuple relational calculus, plus commands to insert, delete and modify tuples.
- **Embedded data manipulation language** - for use within programming languages like C, PL/1, Cobol, Pascal, etc.
- **View Definition** - commands for defining views
- **Authorisation** – this specifies the access rights to relations and views.
- **Integrity** - a limited form of integrity checking.
- **Transaction control** - specifying beginning and end of transactions.

SELF-ASSESSMENT EXERCISE

Describe the concept of an interactive data manipulation language.

4.0 CONCLUSION

Winding up, we can go over the main points of this unit. The Structured Query Language (SQL) is regarded as the standard relational database language. A typical SQL consists of several parts which include: Data definition language (DDL), Interactive data manipulation language (DML), Embedded data manipulation language, View Definition, Authorisation, Integrity and Transaction control. The Data definition language (DDL) - provides commands to: Define relation schemes, Delete relations, Create indices and Modify schemes. The Interactive data manipulation language (DML) - a query language based on both relational algebra and tuple relational calculus, plus commands to insert, delete and modify tuples.

5.0 SUMMARY

This unit provided an overview of Structured Query Language (SQL), specifying the structural components of SQL. We hope you have found this unit interesting.

6.0 TUTOR-MARKED ASSIGNMENT

1. List at least four structural components of SQL.
2. State the specific roles of a data definition language.
3. Describe the concept of an interactive data manipulation language.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D). *Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T.(Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?”. *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More.* Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching.* Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). *Database Management Systems* (2nd Ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009). *Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 2 SQL Expressions

CONTENTS

1.0 INTRODUCTION

In the previous unit, we established the structural components of a Structured Query Language (SQL). However, in this unit we will take a closer look at the components of a classical SQL expression.

2.0 OBJECTIVES

At the end this unit, you should be able to:

- state the components of an SQL expression
- explain the key role of each component of an SQL expression
- indicate specific examples of each of the clauses.

3.0 MAIN CONTENT

3.1 Components of an SQL Expression

An SQL expression consists of **select, from** and **where** clauses.
Ordinarily, a query has the form

select $A_1, A_2, \ldots, A_n$

from $r_1, r_2, \ldots, r_m$

where *P*

where each A_i represents an attribute, each r_i a relation, and *P* is a predicate.

This is equivalent to the relational algebra expression

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

If the where clause is omitted, the predicate *P* is true. The list of attributes can equally be replaced with a * to select all. SQL forms the Cartesian product of the relations named, performs a selection using the predicate, then projects the result onto the attributes named. The result of an SQL query is a relation. SQL may internally convert into more efficient expressions.

3.2 The "SELECT" Clause

The **select** clause lists attributes to be copied. It corresponds to relational algebra **project**.

An example of the select clause is as follows: Find the names of all branches in the *account* relation.

select *bname*
from *account*

distinct vs. **all**: elimination or not elimination of duplicates.

Find the names of all branches in the *account* relation.

select distinct *bname*

from *account*

By default, duplicates are not removed. We can state it explicitly using **all**.

select all *bname*

from *account*

When the asterisk is placed after the select clause, i.e. ***select **** this denotes select all the attributes. Arithmetic operations can also be used in the selection list.

SELF-ASSESSMENT EXERCISE

State a typical expression using the **'select'** clause

3.3 The 'Where' Clause

The '***where***' clause corresponds to the selection predicate in a relational algebra. The predicates can be more complicated, and can involve the following:

- Logical connectives **and, or** and **not**.
- Arithmetic expressions on constant or tuple values.
- The **between** operator for ranges of values.
- This theory can be applied in the following example: Find account number of accounts with balances between $90,000 and $100,000.

select *account#*
from *account*
where *balance* **between** *90000* **and** *100000*

3.4 The 'From' Clause

This is the clause that corresponds to Cartesian product, which lists relations to be used. The '**from'** class by itself defines a Cartesian product of the relations in the clause.

SQL does not have a natural join equivalent. However, natural join can be expressed in terms of a Cartesian product, selection, and projection. For the relational algebra expression

$$\Pi_{cname,loan\#}(borrower \bowtie loan)$$

We can represent this by means of an SQL statement as follows:

select distinct *cname, borrower.loan#*

from *borrower, loan*

where *borrower.loan# = loan.loan#*

4.0 CONCLUSION

In conclusion, SQL expression consists of **Select, From and Where** clauses. The select clause lists attributes to be copied. The **'*where*'** clause corresponds to the selection predicate in a relational algebra. While the 'From' clause corresponds to the Cartesian product, which lists relations to be used.

5.0 SUMMARY

In this unit, we treated the components of an SQL expression. We equally specified the syntax for the 'Select', 'Where' and 'From' Clause. We hope that you found this unit enlightening.

6.0 TUTOR-MARKED ASSIGNMENT

1. In the SQL context, how are natural joins expressed?
2. State the function of the following operators: OR, BETWEEN, AND
3. How are duplicates handled in a 'SELECT' clause?

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D). *Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970)."A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

"Database Design Basics". (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). "Is SQL a Standard Anymore?". *TechRepublic's Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. (1st Ed.). Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching.* Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th Ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 3 DATABASE MODIFICATION

CONTENTS

1.0 INTRODUCTION

Up until now, we had looked at the aspect of extracting information from the database. In this unit we shall consider the common modes of database modification.

2.0 OBJECTIVES

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

- identify the common forms of database modification
- state the difference between tuples and relations
- give the general syntax for the 'Deleting, Inserting and Updating' databases.

3.0 MAIN CONTENT

3.1 Modes of Database Modification

An interesting feature of databases is the transformative capacity. We will consider the common forms of a database modification in the ensuing units.

3.1.1 Deletion

Deletion is expressed in much the same way as a query. Instead of displaying, the selected tuples are removed from the database. We can only delete whole tuples.

The syntax for **deletion** in SQL is given as:

delete from *r*
where *P*

Tuples in *r* for which *P* is true are deleted. In the event that the 'where' clause is omitted, all tuples are deleted.

Furthermore, the request **delete from** *loan* deletes **all** tuples from the relation *loan*. Other examples are as follows:

1. Delete all of Smith's account records.
2. **delete from** *depositor*
3. **where** *cname="Smith"*
4. Delete all loans with loan numbers between 1300 and 1500.
5. **delete from** *loan*
6. **where** *loan#* **between** 1300 **and** 1500
7. Delete all accounts at branches located in Surrey.
8. **delete from** *account*
9. **where** *bname* **in**
10. (**select** *bname*
11. **from** *branch*
12. **where** *bcity="Surrey"*)

Tuples can only be deleted from one relation at a time, but we may reference any number of relations in a **select-from-where** clause embedded in the **where** clause of a **delete**.

However, if the delete request contains an embedded select that references the relation from which tuples are to be deleted, ambiguities may result.

For example, to delete the records of all accounts with balances below the average, we might write

delete from *account*

where *balance* <

(**select avg**(*balance*) **from** *account*)

In this case, when we delete tuples from *account*, the average balance changes!

SELF-ASSESSMENT EXERCISE

Identify the main difference between Tuples and Relations.

3.1.2 Insertion

To insert data into a relation, we either ***specify a tuple***, or ***write a query*** whose result is the set of tuples to be inserted. Attribute values for inserted tuples must be members of the attribute's domain.

Some examples:

1. To insert a tuple for Smith who has $1200 in account A-9372 at the SFU branch.
2. **insert into** *account*
3. **values** (*"SFU",* "A-9372", 1200)
4. To provide each loan that the customer has in the SFU branch with a $200 savings account.
5. **insert into** *account*
6. **select** *bname, loan#, 200*
7. **from** *loan*
8. **where** *bname="SFU"*

Here, we use a **select** to specify a set of tuples.

It is important that we evaluate the **select** statement fully before carrying out any insertion. If some insertions were carried out even as the **select** statement was being evaluated, the insertion might insert an infinite number of tuples. Evaluating the **select** statement completely before performing insertions avoids such problems.

insert into *account*
select *
from *account*

It is possible for inserted tuples to be given values on only some attributes of the schema. The remaining attributes are assigned a null value denoted by *null*.

The insertion of null values can be prohibited by using the SQL DDL.

3.1.3 Updates

The 'Update' statement allows us to change some values in a tuple without necessarily changing all. The example below demonstrates this as follows:

1. To increase all account balances by 5 percent.
2. **update** *account*
3. **set** *balance=balance* * 1.05

This statement is applied to every tuple in *account.*

To make two different rates of interest payment, depending on balance amount:

1. **update** *account*
2. **set** *balance=balance* * 1.06
3. **where** *balance* > 10,000
4. **update** *account*
5. **set** *balance=balance* * 1.05
6. **where** *balance* $\leq$ 10,000

4.0 CONCLUSION

To wrap up, tuples can only be deleted from one relation at a time. Data is inserted into a relation by either specifying a tuple, or writing a query whose result is the set of tuples to be inserted. Selective alteration of tuples is made possible by means of 'Update' statements.

5.0 SUMMARY

We considered the common forms of database modification, specifying the general syntax of some common database statement. To test your knowledge, attempt the exercise below.

6.0 TUTOR-MARKED ASSIGNMENT

1. In the context of database modification, state the main difference between Tuples and Relations
2. State the general syntax for deleting databases

7.0 REFERENCES/FURTHER READING

Avi, S; *et al*. (N.D).*Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?”. *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL* (1st Ed.). Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). “Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus”. *Journal of Computers in Mathematics and Science Teaching* ISSN 0731-

9258 0731-9258 Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009). *Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 4 INTEGRITY CONSTRAINTS

CONTENTS

1.0 INTRODUCTION

In this unit we will learn about the concept of Integrity constraints. We would also be taught two core aspects of referential integrity: referential integrity in the E-R Model and referential integrity in SQL. Hope you would be able to grasp the main ideas.

2.0 OBJECTIVES

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

- define domain constraints
- show the link between main Constraints and Integrity Constraints
- distinguish between referential integrity in the E-R Model and referential integrity in SQL.

3.0 MAIN CONTENT

3.1 Domain Constraints

A domain constraint specifies the permissible values for a given attribute, while a key constraint specifies the attributes that uniquely identify a row in a given table.

3.1.1 Domain/Key Normal Form (DKNF)

Domain/key normal form (DKNF) is a normal form used in database normalisation which requires that the database contains no constraints other than domain constraints and key constraints.

It is required in databases to prevent the occurrence of general constraints in the database that are not clear domain or key constraints.

3.1.2 Domain Constraints and Integrity Constraints

A domain of possible values should be associated with every attribute. These domain constraints are the most basic form of integrity constraints. They are easy to test for when data is entered.

3.1.3 Domain Constraint Guidelines

There are a number of guidelines to adopt in implementing domain constraints. The key points are itemised below as follows:

- Attributes may have the same domain, e.g. *cname* and *employee-name*.
- It is not as clear whether *bname* and *cname* domains ought to be distinct.
- At the implementation level, they are both character strings
- At the conceptual level, we do not expect customers to have the same names as branches, in general
- Strong typing of domains enables one to test for values inserted, and whether queries make sense. Newer systems, particularly object-oriented database systems, offer a rich set of domain types that can be extended easily

3.2 The Check Clause

The check clause in SQL-92 permits domains to be restricted in powerful ways that most programming language type systems do not permit.

Furthermore, the check clause enables schema designer specify a predicate that must be satisfied by any value assigned to a variable whose type is the domain.

Examples: create domain *hourly-wage* numeric (5,2)
constraint *wage-value-test* check(value >= 4.00)

Note that "constraint *wage-value-test*" is optional (to give a name to the test to signal which constraint is violated).

create domain *account-number* char(10)

constraint *account-number-null-test*
check(value not null)

create domain *account-type* char(10)

constraint *account-type-test*
check(value in ("Checking"', "Saving"))

3.3 Referential Integrity

Often we wish to ensure that a value appearing in a relation for a given set of attributes also appears for another set of attributes in another relation.

This is called referential integrity.

3.3.1 Referential Integrity in the E-R Model

These constraints arise frequently. Every relation arising from a relationship set has referential integrity constraints.

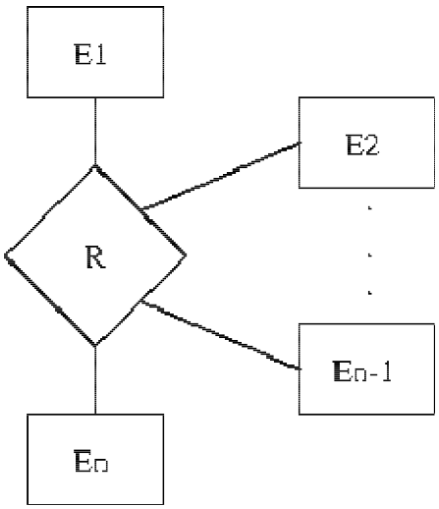

Fig. 3.1: An n-ary Relationship Set

Fig. 3.1 Shows an n-ary relationship set *R* relating

entity sets $E_1, E_2, \ldots E_n$.

Let K_i denote the primary key of E_i.

The attributes of the relation scheme for relationship

set *R* include $K_1 \cup K_2 \cup \ldots \cup K_n$.

Each K_i in the scheme for *R* is a foreign key that leads to a referential integrity constraint.

Relation schemes for weak entity sets must include the primary key of the strong entity set on which they are existence dependent. This is a foreign key, which leads to another referential integrity constraint.

3.3.2 Referential Integrity in SQL

An addition to the original standard allows specification of primary and candidate keys and foreign keys as part of the create table command:

- primary key clause includes a list of attributes forming the primary key.
- unique key clause includes a list of attributes forming a candidate key.
- foreign key clause includes a list of attributes forming the foreign key,
- and the name of the relation referenced by the foreign key.

The example below illustrates a summary of the features mentioned so far:

create table *customer*

(*cname* char(20) not null,

street char(30),

city char(30),

primary key (*cname*))

create table *branch*

(*bname* char(15) not null,

bcity char(30),

assets integer,

primary key (*bname*)

check (*assets* >= 0))

create table *account*

(*account#* char(10) not null,

(*bname* char(15),

balance integer,

primary key (*account#*)

foreign key (*bname*) references *branch*,

check (*balance* >= 0))

create table *depositor*

(*cname* char(20) not null,

account# char(10) not null,

primary key (*cname, account#*)

foreign key (*cname*) references *customer*,

foreign key (*account#*) references *account*)

SELF-ASSESSMENT EXERCISE

What are the components of a unique key?

3.4 Foreign Keys

Fundamentally, in the database context, a foreign key simply refers to the short form for declaring a single column. For example: *bname* **char**(15) **references** *branch*

Normally, when a referential integrity constraint is violated, the action is rejected.

However, a foreign key clause in SQL-92 can specify steps to be taken to change the tuples in the referenced relation to restore the constraint.

For example:

create table *account*

...

foreign key (*bname*) references *branch*

on delete cascade

on insert cascade,

...

If a delete of a tuple in *branch* results in the preceding referential integrity constraints being violated, the delete is not rejected, but the delete "cascade" to the *account* relation, deleting the tuple that refers to the branch that was deleted.

Update will be cascaded to the new value of the branch!

SQL-92 also allows the foreign key clause to specify actions other than cascade, such as setting the referencing field to null, or to a default value, if the constraint is violated.

If there is a chain of foreign key dependencies across multiple relations, a deletion or update at one end of the chain can propagate across the entire chain.

If a cascading update or delete causes a constraint violation that cannot be handled by a further cascading operation, the system aborts the transaction and all the changes caused by the transaction and its cascading actions are undone.

Given and complexity and arbitrary nature of the way constraints in SQL behave with null values, it is the best to ensure that all columns of unique and foreign key specifications are declared to be non-null.

4.0 CONCLUSION

In this unit, we discovered that permissible values for a given attribute are specified by the domain constraint, while a key constraint specifies the attributes that uniquely identify a row in a given table. We also considered the fact that the Domain/Key Normal form is required in databases, to prevent the occurrence of general constraints in the database that are not clear domain or key constraints. The 'check' clause enables schema designer specify a predicate that must be satisfied by any value assigned to a variable whose type is the domain. The fact that strong typing of domains enables one test for values to be inserted was highlighted. In the database context, a foreign key simply refers to the short form for declaring a single column.

5.0 SUMMARY

This unit introduced the concept of domain constraints and integrity constraints. The 'Check' clause and Referential integrity were equally highlighted. We hope you enjoyed this unit.

6.0 TUTOR-MARKED ASSIGNMENT

1. What is the key role of the domain constraint?
2. State the significance of the domain/key normal form with respect to databases?
3. Mention the main purpose of the schema designer.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D). *Database System Concepts* (6th ed.). McGraw-Hill.

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E.F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E.F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?” *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). “Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus”. *Journal of Computers in Mathematics and Science Teaching*. Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). “Referential Integrity”. http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM,* 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

MODULE 4 COMPUTER DATA STORAGE AND FILE STRUCTURE

UNIT 1 COMPUTER DATA STORAGE AND LEVELS

CONTENTS

1.0 INTRODUCTION

In the previous modules, we considered domain constraints, integrity constraints and referential integrity with respect to Structured Query Language (SQL). This unit presents the concept of computer data storage, the different levels of storage and corresponding forms of storage.

2.0 OBJECTIVES

At the end this unit, you should be able to:

- describe computer data storage
- classify the levels of storage
- itemise the fundamental components of a basic computer
- differentiate between the auxiliary and main memory
- state the implication of the volatility of the primary storage being volatile.

3.0 MAIN CONTENT

3.1 Computer Data Storage

Computer data storage, often called **storage** or **memory**, refers to computer components and recording media that retain digital data used for computing for some interval of time. Computer data storage provides one of the core functions of the modern computer, that of information retention. It is one of the fundamental components of all modern computers, and coupled with a central processing unit (CPU, a processor), implements the basic computer model.

3.2 Levels of Storage

There are different forms of storage, divided according to their distance from the central processing unit. The fundamental components of a general-purpose computer are arithmetic and logic unit, control circuitry, storage space, and input/output devices.

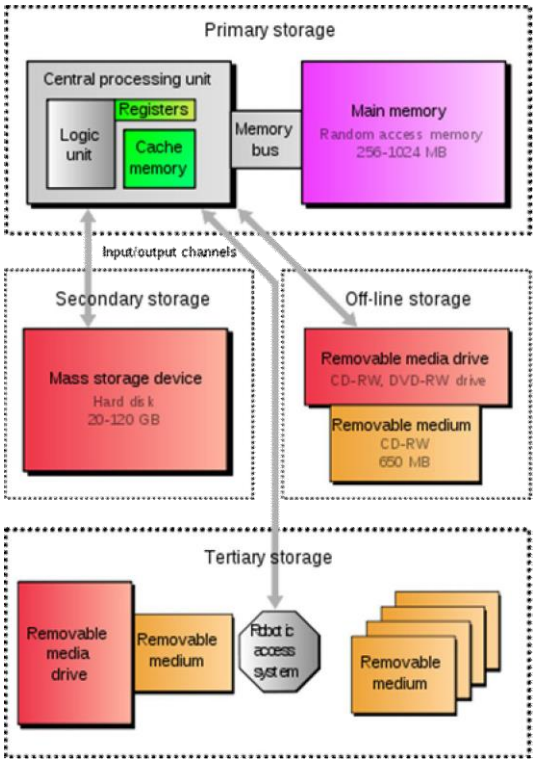

Fig. 4.1: Levels of Storage

3.2.1 Primary Storage

Primary storage (or **main memory** or **internal memory**), commonly referred to as **memory**, is the only memory directly accessible to the CPU. The CPU continuously reads instructions stored there and executes them as required. Any data actively operated on is also stored there in uniform manner.

Historically, early computers used rotating magnetic drums as primary storage. These were later replaced by magnetic core memory and subsequently by integrated circuit. This led to modern random-access memory (RAM). It is small-sized, light, but quite expensive at the same time. (The particular types of RAM used for primary storage are also volatile, i.e. they lose the information when not powered).

Main memory is directly or indirectly connected to the central processing unit via a *memory bus*. This consists of two buses: an address bus and a data bus. The CPU initially sends a number through an address bus. This number called the memory address indicates the desired location of data. Then it reads or writes the data itself using the data bus.

As the RAM types used for primary storage are volatile (cleared at start up), a computer containing only such storage would not have a source to read instructions from, in order to start the computer. Hence, non-volatile primary storage containing a small startup program (BIOS) is used to bootstrap the computer, that is, to read a larger program from non-volatile *secondary* storage to RAM and start to execute it. A non-volatile technology used for this purpose is called ROM, for read-only memory (the terminology may be somewhat confusing as most ROM types are also capable of *random access*).

Many types of ‘ROM’ are not literally *read only*, as updates are possible; however it is slow and memory must be erased in large portions before it can be re-written. Some embedded systems run programs directly from ROM (or similar), because such programs are rarely changed. Standard computers do not store non-rudimentary programs in ROM, rather use large capacities of secondary storage, which is non-volatile as well, and not as costly.

3.2.2 Secondary Storage

Secondary storage (also known as external memory or auxiliary storage), differs from primary storage in that it is not directly accessible by the CPU. The computer usually uses its input/output channels to access secondary storage and transfers the desired data using intermediate area in primary storage. Secondary storage does not lose the data when the device is powered down—it is non-volatile. Per unit, it is typically also two orders of magnitude less expensive than primary storage. Consequently, modern computer systems typically have two orders of magnitude, more secondary storage than primary storage and data is kept for a longer time there.

3.2.3 Types of Secondary Storage

Hard disk drives

In modern computers, hard disk drives are usually used as secondary storage. The time taken to access a given byte of information stored on a hard disk is typically a few thousandths of a second, or milliseconds. By contrast, the time taken to access a given byte of information stored in random access memory is measured in billionths of a second, or nanoseconds. This illustrates the significant access-time difference which distinguishes solid-state memory from rotating magnetic storage devices: hard disks are typically about a million times slower than memory.

Fig. 4.2: A Hard Disk Drive with Protective Cover Removed

Rotating optical storage devices such as CD and DVD drives have even longer access times. With disk drives, once the disk read/write head reaches the proper placement and the data of interest rotates under it, subsequent data on the track are very fast to access. As a result, in order to hide the initial seek time and rotational latency, data is transferred to and from disks in large contiguous blocks.

Some other examples of secondary storage technologies are: flash memory (e.g. USB flash drives or keys), floppy disks, magnetic tape, paper tape, punched cards, standalone RAM disks, and Iomega Zip drives.

Normally, the secondary storage is often formatted according to a file system format, which provides the abstraction necessary to organise data into files and directories, providing also additional information (called metadata) which describes the owner of a certain file, the access time, the access permissions, and other information.

Most computer operating systems use the concept of virtual memory, allowing utilisation of more primary storage capacity than is physically available in the system. As the primary memory fills up, the system moves the least-used chunks (*pages*) to secondary storage devices (to a swap file or page file), retrieving them later when they are needed. As more of these retrievals from slower secondary storage are necessary, the more the overall system performance is degraded.

SELF-ASSESSMENT EXERCISE

List at least three common types of secondary storage.

3.2.4 Tertiary Storage

Tertiary storage or **tertiary memory** provides a third level of storage. Typically, it involves a robotic mechanism which will *mount* (insert) and *dismount* removable mass storage media into a storage device according to the system's demands; this data is often copied to secondary storage before use. It is primarily used for archiving rarely accessed information since it is much slower than secondary storage (e.g. 5–60 seconds vs. 1-10 milliseconds). This is primarily useful for extraordinarily large data stores, accessed without human operators. Typical examples include tape libraries and optical jukeboxes.

Fig. 4.3: Large Tape Library

Tape cartridges placed on shelves in the front, robotic arm moving in the back. Visible height of the library is about 180 cm.

When a computer needs to read information from the tertiary storage, it will first consult a catalog database to determine which tape or disc contains the information. Next, the computer will instruct a robotic arm to fetch the medium and place it in a drive. When the computer has finished reading the information, the robotic arm will return the medium to its place in the library.

3.2.5 Off-line Storage

Off-line storage is computer data storage on a medium or a device that is not under the control of a processing unit. The medium is recorded, usually in a secondary or tertiary storage device, and then physically removed or disconnected. It must be inserted or connected by a human operator before a computer can access it again. Unlike tertiary storage, it cannot be accessed without human interaction.

This storage is used to transfer information, since the detached medium can be easily physically transported. Additionally, in case a disaster, for example a fire, destroys the original data, a medium in a remote location will probably be unaffected, enabling disaster recovery. Off-line storage increases general information security, since it is physically inaccessible from a computer, and data confidentiality or integrity cannot be affected by computer-based attack techniques. Also, if the information stored for

archival purposes is accessed seldom or never, off-line storage is less expensive than tertiary storage.

In modern personal computers, most secondary and tertiary storage media are also used for off-line storage. Optical discs and flash memory devices are most popular, and to much lesser extent removable hard disk drives. In enterprise uses, magnetic tape is predominant. Older examples are floppy disks, Zip disks, or punched cards.

4.0 CONCLUSION

On the whole, we learnt that Computer data storage refers to computer components and recording media that retain digital data used for computing for some interval of time. There are different forms of storage, divided according to their distance from the central processing unit. The fundamental components of a general-purpose computer are arithmetic and logic unit, control circuitry, storage space, and input/output devices. The main memory is the only memory directly accessible to the CPU. It is directly or indirectly connected to the central processing unit via a *memory bus*. The CPU continuously reads instructions stored in the main memory and executes them as required.

On the other hand, the Secondary storage is not directly accessible by the CPU. Common forms of Secondary Storage include: hard disk drives, rotating optical storage devices, flash memory, floppy disks, magnetic tape, paper tape, punched cards, standalone RAM disks, Iomega Zip drives and a host of others. Most computer operating systems use the concept of virtual memory. The tertiary storage or memory involves a robotic storage mechanism which mounts and dismounts removable mass storage media into a storage device according to the system's demands. We equally identified the off-line storage as a form of computer data storage on a device that is not under the control of a processing unit. This storage is used to transfer information, since the detached medium can be easily physically transported.

5.0 SUMMARY

In this unit, we learnt about computer data storage and the levels of storage as well as classical examples of each form of storage. Now to discover if you have been following, please answer the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. What is the core distinction between the auxiliary and internal memory?
2. Explain the implication of the volatility of the primary storage being volatile.
3. In the data context, list the levels of storage.

7.0 REFERENCES/FURTHER READING

Avi, S. *et al*. (N.D). *Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?” *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching.* Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). *Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design*. (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 2 FEATURES OF STORAGE TECHNOLOGIES

CONTENTS

1.0 INTRODUCTION

In this unit, you will be taught the core aspects of storage technologies. These include: volatility, mutability, accessibility, and addressability. However, for the implementation of any storage technology, the characteristics worth measuring are capacity and performance. Keep these ideas in mind and do enjoy your studies.

2.0 OBJECTIVES

At the end this unit, you should be able to:

- outline the core characteristics of storage technologies
- differentiate between the volatile and non-volatile memory
- explain the mechanism of the Read/Write storage
- distinguish between the Read/Write and Read only storage
- describe the Slow write, fast read storage
- identify the forms of accessibility, addressability
- explain the notion of latency
- describe the term ‘throughput’.

3.0 MAIN CONTENT

3.1 Volatility

3.1.1 Non-Volatile Memory

This is a form of memory that retains stored information even if it is not constantly supplied with electric power. It is suitable for long-term storage of information.

3.1.2 Volatile Memory

This form of memory requires constant power to maintain the stored information. The fastest memory technologies of today are volatile ones (not a universal rule). Since primary storage is required to be very fast, it predominantly uses volatile memory.

3.2 Differentiation

3.2.1 Dynamic Random Access Memory

A form of volatile memory which also requires the stored information to be periodically re-read and re-written, or refreshed, otherwise it would vanish.

3.2.2 Static Memory

A form of volatile memory similar to DRAM, that never needs to be refreshed as long as power is applied. (It loses its content if power is removed).

SELF-ASSESSMENT EXERCISE

State the major difference between the volatile and non-volatile memory.

3.3 Mutability

3.3.1 Read/Write Storage or Mutable Storage

This form of storage allows information to be overwritten at any time. A computer without some amount of read/write storage for primary storage purposes would be useless for many tasks. Modern computers typically use read/write storage also for secondary storage.

3.3.2 Read Only Storage

Retains the information stored at the time of manufacture, and write once storage (Write Once Read Many) allows the information to be written only once at some point after manufacture. These are called immutable storage. Immutable storage is used for tertiary and off-line storage. Examples include CD-ROM and CD-R.

3.3.3 Slow Write, Fast Read Storage

Read/write storage which allows information to be overwritten multiple times, but with the write operation being much slower than the read operation. Examples include CD-RW and flash memory.

3.4 Accessibility

This feature can be categorised in two ways:

3.4.1 Random Access

Any location in storage can be accessed at any moment in approximately the same amount of time. Such characteristic is well suited for primary and secondary storage.

3.4.2 Sequential Access

The accessing of pieces of information will be in a serial order, one after the other; therefore the time to access a particular piece of information depends upon which piece of information was last accessed. Such characteristic is typical of off-line storage.

3.5 Addressability

3.5.1 Location-Addressable

Each individually accessible unit of information in storage is selected with its numerical memory address. In modern computers, location-addressable storage usually limits to primary storage, accessed internally by computer programs, since location-addressability is very efficient, but burdensome for humans.

3.5.2 File-Addressable

Information is divided into *files* of variable length, and a particular file is selected with human-readable directory and file names. The underlying device is still location-addressable, but the operating system of a computer provides the file system abstraction to make the operation more understandable. In modern computers, secondary, tertiary and off-line storage use file systems.

3.5.3 Content-Addressable

Each individually accessible unit of information is selected based on the basis of (part of) the contents stored there. Content-addressable storage can be implemented using software (computer program) or hardware (computer device), with hardware being faster but more expensive option. Hardware content addressable memory is often used in a computer's CPU cache.

3.6 Capacity

3.6.1 Raw Capacity

This is the total amount of stored information that a storage device or medium can hold. It is expressed as a quantity of bits or bytes (e.g. 10.4 megabytes).

3.6.2 Memory Storage Density

This refers to the compactness of stored information. It is the storage capacity of a medium divided with a unit of length, area or volume (e.g. 1.2 megabytes per square inch).

3.7 Performance

3.7.1 Latency

The time it takes to access a particular location in storage. The relevant unit of measurement is typically nanosecond for primary storage, millisecond for secondary storage, and second for tertiary storage. It may make sense to separate read latency and write latency, and in case of sequential access storage, minimum, maximum and average latency.

3.7.2 Throughput

The term 'throughput' simply refers to the rate at which information can be read from or written to the storage. In computer data storage, throughput is usually expressed in terms of megabytes per second or MB/s, though bit rate may also be used. As with latency, read rate and write rate may need to be differentiated. Also accessing media sequentially, as opposed to randomly, typically yields maximum throughput.

3.8 Energy Use

Storage devices that reduce fan usage, automatically shut-down during inactivity, and low power hard drives can reduce energy consumption 90 percent. 2.5 inch hard disk drives often consume less power than larger ones. Low capacity solid-state drives have no moving parts and consume less power than hard disks. Also, memory may use more power than hard disks.

4.0 CONCLUSION

To wrap up, we discovered that while the volatile memory requires constant power to maintain the stored information is the volatile memory, non-volatile memory retains stored information even if it is not constantly supplied with electric power. Modern computers typically use read/write storage also for secondary storage. We also learnt that the time it takes to access a particular location in storage is referred to as ***latency***.

5.0 SUMMARY

In this unit, we were able to distinguish between the volatile and non-volatile memory, explain the mechanism of the Read/Write storage **Identify the difference between the Read/Write and Read only storage.**

Describe the Slow write; fast read storage, Identify the forms of accessibility, addressability. Explain the notion of latency as well as explain the term 'throughput'.

6.0 TUTOR-MARKED ASSIGNMENT

1. Identify the difference between the Read/Write and Read only storage.
2. Describe the Slow write, fast read storage.
3. Identify the forms of accessibility, addressability.
4. Explain the notion of latency.
5. Give a succinct description of the term 'throughput'.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D).*Database System Concept* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970)."A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

"Database Design Basics". (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). "Is SQL a Standard Anymore?". *TechRepublic's Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993). *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching* .Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009). *Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models*. Englewood-Cliffs: Prentice-Hall.

UNIT 3 COMMON STORAGE TECHNOLOGIES

CONTENTS

1.0 INTRODUCTION

In this unit, you will learn the different forms of storage technologies. We will consider specific examples and some real-life applications. Do enjoy your studies.

2.0 OBJECTIVES

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

- explain the storage mechanism of semi conductor memory
- describe the storage mechanism of the magnetic storage, optical disc and the magneto-optical disc storage
- differentiate between the optical disc and the magneto-optical disc storage
- cite common examples of optical disc storage
- mention the regular forms of uncommon storage.

3.0 MAIN CONTENT

3.1 Semi Conductors

Semi conductor memory uses semi conductor-based integrated circuits to store information. A semiconductor memory chip may contain millions of tiny transistors or capacitors. Both *volatile* and *non-volatile* forms of semi conductor memory exist. In modern computers, primary storage almost exclusively consists of dynamic volatile semi conductor

memory or dynamic random access memory. Since the turn of the century, a type of non-volatile semiconductor memory known as flash memory has steadily gained share as off-line storage for home computers. Non-volatile semiconductor memory is also used for secondary storage in various advanced electronic devices and specialised computers.

3.2 Magnetic Storage

Magnetic storage uses different patterns of magnetisation on a magnetically coated surface to store information. Magnetic storage is *non-volatile*. The information is accessed using one or more read/write heads which may contain one or more recording transducers. A read/write head only covers a part of the surface so that the head or medium or both must be moved relative to another in order to access data. In modern computers, magnetic storage will take these forms:

3.2.1 Magnetic Disk

- Floppy disk, used for off-line storage
- Hard disk drive, used for secondary storage

3.2.2 Magnetic Tape Data Storage (Used for Tertiary and Off-Line Storage)

Previously, magnetic storage was also used for primary storage in a form of magnetic drum, or core memory, core rope memory, thin-film memory, twistor memory or bubble memory. Also unlike today, magnetic tapes are frequently used for secondary storage.

SELF-ASSESSMENT EXERCISE

Give a brief description of the magnetic storage mechanism.

3.3 Optical Storage

A typical optical disc, stores information on the surface of a circular disc and reads this information by illuminating the surface with a laser diode and observing the reflection. Optical disc storage is *non-volatile*. The deformities may be permanent (read only media), formed once (write once media) or reversible (recordable or read/write media). The following forms are currently in common use:

- CD, CD-ROM, DVD, BD-ROM: Read only storage, used for mass distribution of digital information (music, video, computer programs)

- CD-R, DVD-R, DVD+R, BD-R: Write once storage, used for tertiary and off-line storage
- CD-RW, DVD-RW, DVD+RW, DVD-RAM, BD-RE: Slow write, fast read storage, used for tertiary and off-line storage
- Ultra Density Optical or UDO is similar in capacity to BD-R or BD-RE and is slow write, fast read storage used for tertiary and off-line storage.

3.4 Magneto-Optical Disc Storage

This is a form of optical disc storage where the magnetic state on a ferromagnetic surface stores information. The information is read optically and written by combining magnetic and optical methods. Magneto-optical disc storage is *non-volatile*, *sequential access*, slow write, fast read storage used for tertiary and off-line storage.

3.5 Paper Data Storage

This storage is typically in the form of paper tape or punched cards, has long been used to store information for automatic processing, particularly before general-purpose computers existed. Information was recorded by punching holes into the paper or cardboard medium and was read mechanically (or later optically) to determine whether a particular location on the medium was solid or contained a hole. A few technologies allow people to make marks on paper that are easily read by machine—these are widely used for tabulating votes and grading standardised tests. Barcodes made it possible for any object that was to be sold or transported to have some computer readable information securely attached to it.

3.6 Uncommon Storage

3.6.1 Vacuum Tube Memory

A Williams tube used a cathode ray tube, and a Selectron tube used a large vacuum tube to store information. These primary storage devices were short-lived in the market, since Williams tube was unreliable and Selectron tube was expensive.

3.6.2 Electro-Acoustic Memory

Delay line memory used sound waves in a substance such as mercury to store information. Delay line memory was dynamic volatile, cycle sequential read/write storage, and was used for primary storage.

3.6.3 Optical Tape

This is a medium for optical storage generally consisting of a long and narrow strip of plastic onto which patterns can be written and from which the patterns can be read back. It shares some technologies with cinema film stock and optical discs, but is compatible with neither. The motivation behind developing this technology was the possibility of far greater storage capacities than either magnetic tape or optical discs.

4.0 CONCLUSION

In conclusion, we have seen that at the moment, the most commonly used data storage technologies are semiconductor, magnetic, and optical, while paper still finds some limited usage. Some other fundamental storage technologies have also been used in the past or are proposed for development.

5.0 SUMMARY

In sum, we discovered the common types of data storage technologies. We equally considered the uncommon storage technologies. You can now attempt the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. Describe the storage mechanism of the magneto-optical disc storage.
2. State the core difference between the optical disc and the magneto-optical disc storage.
3. List common examples of optical disc storage.
4. Identify the regular forms of uncommon storage.
5. Cite at least three examples of optical disc storage.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D).*Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?”. *TechRepublic's Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). “Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus”. *Journal of Computers in Mathematics and Science Teaching* .Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). “Referential Integrity”. http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM,* 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 4 FILE ORGANISATION

CONTENTS

1.0 INTRODUCTION

This unit presents key considerations in specifying a system of file organisation as well as techniques of file organisation. You will actually find this aspect simple and interesting. Enjoy your studies!

2.0 OBJECTIVES

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

- explain the concept of file organisation
- identify the components of file organisation
- state the key considerations in specifying a system of file organisation
- outline the common methods of organising files.

3.0 MAIN CONTENT

3.1 Introduction to File Organisation

File organisation is the methodology which is applied to structured computer files. In general, files contain computer records which can be documents or information which is stored in a certain way for later retrieval.

Consequently, **file organisation** primarily refers to the logical arrangement of data (which can itself be organised in a system of records with correlation between the fields/columns) in a file system. It should not be confused with the physical storage of the file in some types of storage media. There are certain basic types of computer file, which can include files stored as blocks of data and streams of data, where the information streams out of the file while it is being read until the end of the file is encountered.

Two significant components of file organisation are:

- The way the internal file structure is arranged and
- The external file as it is presented to the operating system or program that calls it.

3.2 Methods and Design Paradigm

The design of the file organisation depends mainly on the system environment. For instance, factors such as whether the file is going to be used for *transaction-oriented processes* like OLTP or Data Warehousing, or whether the file is *shared among various processes* like those found in a typical distributed system or standalone. Whether the file is on a network and used by a number of users and whether it may be accessed internally or remotely and how often it is accessed must also be determined.

3.3 System File Organisation Specifics

The task of specifying a system of file organisation for a computer software program or a computer system designed for a particular purpose is a high-level design decision. Therefore, the key considerations in specifying a system of file organisation are as follows:

- Rapid access to a record or a number of records which are related to each other.
- The adding, modification, or deletion of records.
- Efficiency of storage and retrieval of records.
- Redundancy, being the method of ensuring data integrity.

Thus, a file should be organised in such a way that the records are always available for processing with no delay. This should be done in line with the activity and volatility of the information.

3.4 Factors that affect File Organisation

Organising a file depends on what kind of file it happens to be: a file in the simplest form can be a *text file*, (in other words a file which is composed of ASCII (American Standard Code for Information Interchange) text.) Files can also be created as *binary* or *executable* types (containing elements other than plain text.) Also, files are keyed with attributes which help determine their use by the host operating system.

SELF-ASSESSMENT EXERCISE

Mention two important components of file organization.

3.5 File Organisation Technique

The common methods of organising files are:

a. Sequential
b. Line-sequential
c. Indexed-sequential
d. Inverted list
e. Direct or hashed access organisation.

3.5.1 Sequential Organisation

A sequential file contains records organised in the order they were entered. The order of the records is fixed. The records are stored and sorted in physical, contiguous blocks within each block the records are in sequence.

Records in these files can only be read or written sequentially. Once stored in the file, the record cannot be made shorter, or longer, or deleted. However, the record can be *updated* if the length does not change. (This is done by replacing the records by creating a new file.) New records will always appear at the end of the file.

If the **order of the records** in a file is not important, **sequential organisation** will suffice, no matter how many records you may have. Sequential output is also useful for report printing or *sequential reads* which some programs prefer to do.

3.5.2 Line-Sequential Organisation

Line-sequential files are similar to sequential files, except that the records can contain only characters as data. Line-sequential files are maintained by the native byte stream files of the operating system.

In the COBOL environment, line-sequential files that are created with WRITE statements with the ADVANCING phrase can be directed to a printer as well as to a disk.

3.5.3 Indexed-Sequential Organisation

Key searches are improved by means of the indexed-sequential file organisation. The single-level indexing structure is the simplest one where a file, whose records are pairs, contains a key pointer. This *pointer* is the position in the data file of the record with the given key. A subset of the records, which are evenly spaced along the data file, is indexed, in order to mark intervals of data records.

In an indexed-sequential organisation, a key is performed as follows:

- the search key is compared with the index keys to find the highest index key coming in front of the search key
- simultaneously a linear search is performed from the record that the index key points to, until the search key is matched or until the record pointed to by the next index entry is reached.

Regardless of double file access (index + data) required by this sort of search, the access time reduction is significant compared with sequential file searches.

3.5.4 Inverted List

In an inverted list file organisation, the file is indexed on many of the attributes of the data itself. The inverted list method has a single index for each key type. The records are not necessarily stored in a sequence. They are placed in the data storage area, while indexes are updated for the record keys and location.

Here is an example, in a company file, an index could be maintained for all *products*, and another one might be maintained for *product types*. Thus, it is faster to search the indexes than every record. These types of files are also known as **"inverted indexes."** Nevertheless, **inverted list files** use more media space and the storage devices get full quickly with this type of organisation. The benefits are apparent immediately because searching is fast. However, updating is much slower.

Content-based queries in text retrieval systems use **inverted indexes** as their preferred mechanism. Data items in these systems are usually stored *compressed* which would normally slow the retrieval process, but the compression algorithm will be chosen to support this technique.

3.5.5 Direct or Hashed Access

In a **direct or hashed access** file organisation, a portion of the disk space is reserved and a "hashing" algorithm computes the record address. So there is additional space required for this kind of file in the store. Records are placed randomly throughout the file. Records are accessed by addresses that specify their disc location. Also, this type of file organisation requires a disk storage rather than tape. It has an excellent search retrieval performance, but care must be taken to maintain the indexes. If the indexes become corrupt, what is left may as well go to the bit-bucket. For this reason, it is critical to have regular backup of this kind of file just as it is for all stored valuable data.

4.0 CONCLUSION

To wrap up, in this unit we discovered that file organisation primarily refers to the logical arrangement of data in a file system. Two significant components of file organisation are: the way the internal file structure is arranged and the external file as it is presented to the operating system or program that calls it. The design of the file organisation depends mainly on the system environment. Other design consideration include: whether the file is on a network and used by a number of users and whether it may be accessed internally or remotely and how often it is accessed must also be determined. The key considerations in specifying a system of file organisation are as follows: Rapid access to a record or a number of records which are related to each other; the adding, modification, or deletion of records; Efficiency of storage and retrieval of records; Redundancy, being the method of ensuring data integrity.

In sum, a file should be organised in such a way that the records are always available for processing with no delay. Organising a file depends on what kind of file it happens to be. The common methods of organising files are: Sequential, Line-sequential, Indexed-sequential, Inverted list and Direct or hashed access organisation.

5.0 SUMMARY

In this unit, we learnt about file organisation techniques and the factors affecting file organisation. By now, you must certainly be ready to answer the questions below.

6.0 TUTOR-MARKED ASSIGNMENT

1. Briefly describe the concept of file organization.
2. What factors affect the core design considerations.
3. State the key considerations in specifying a system of file organization.
4. Mention at least four methods of organising files

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D). *Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S.(2002). “Is SQL a Standard Anymore?”. *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching* . Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009). *Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

MODULE 5 INTRODUCTION TO XML AND WEB SERVICES

UNIT 1 FUNDAMENTALS OF XML

CONTENTS

1.0 INTRODUCTION

This unit presents an introduction to the Extensible Markup Language (XML), at a reasonably technical level in order to gain more insight on the subject of structured documents. In addition to covering the XML 1.0 Specification, this unit equally underscores related XML specifications, which are evolving. Do take note of these key points.

2.0 OBJECTIVES

At the end this unit, you should be able to:

- give a brief description of an XML
- describe the common concepts of XML
- identify the components of a Structured Document
- distinguish between XML and HTML
- state the difference between XML and SGML.

3.0 MAIN CONTENT

3.1 What is XML?

XML is an extensible markup language for documents containing structured information.

Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure.

A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

SELF-ASSESSMENT EXERCISE

State the core components of a structured document.

3.2 Common Concepts of XML

3.2.1 Documents Concept

The number of applications currently being developed that are based on, or make use of, XML documents is truly amazing (particularly when you consider that XML is not yet a year old)! For our purposes, the word “document” refers not only to traditional documents, like this one, but also to the myriad of other XML “data formats”. These include vector graphics, e-commerce transactions, mathematical equations, object meta-data, server APIs, and other kinds of structured information.

3.2.2 XML and HTML

In HTML, both the tag semantics and the tag set are fixed. An <h1> is always a first level heading and the tag <ati.product.code> is meaningless. The W3C, in conjunction with browser vendors and the WWW community, is constantly working to extend the definition of HTML to allow new tags to keep pace with changing technology and to bring variations in presentation (stylesheets) to the Web. However, these changes are always rigidly confined by what the browser vendors have implemented and by the fact that backward compatibility is paramount. And for people who want to disseminate information widely, features

supported by only the latest releases of Netscape and Internet Explorer are not useful.

On the other hand, XML specifies neither semantics nor a tag set. In fact XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there is no predefined tag set, there cannot be any preconceived semantics. All of the semantics of an XML document will either be defined by the applications that process them or by stylesheets.

3.2.3 XML and SGML

While XML is defined as an application profile of SGML, SGML is the Standard Generalised Markup Language defined by ISO 8879. SGML has been the standard, vendor-independent way to maintain repositories of structured documentation for more than a decade, but it is not well suited to serving documents over the web (for a number of technical reasons beyond the scope of this material). Defining XML as an application profile of SGML means that any fully conformant SGML system will be able to read XML documents. However, using and understanding XML documents *does not* require a system that is capable of understanding the full generality of SGML. XML is, roughly speaking, a restricted form of SGML.

For technical purists, it is important to note that there may also be subtle differences between documents as understood by XML systems and SGML systems. In particular, treatment of white space immediately adjacent to tags may be different.

4.0 CONCLUSION

We learnt that XML is an extensible markup language for documents containing structured information. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays. Almost all documents have some structure. A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

The term “document” refers not only to traditional documents, but also to the myriad of other XML “data formats”. These include vector graphics, e-commerce transactions, mathematical equations, object meta-data, server APIs, and other kinds of structured information.

In HTML, both the tag semantics and the tag set are fixed. On the other hand, XML specifies neither semantics nor a tag set. In fact XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there is no predefined tag set, there cannot be any preconceived semantics. All of the semantics of an XML document will either be defined by the applications that process them or by stylesheets. While XML is defined as an application profile of SGML, SGML is the Standard Generalised Markup Language defined by ISO 8879. SGML has been the standard, vendor-independent way to maintain repositories of structured documentation for more than a decade, but it is not well suited to serving documents over the web. Thus, XML is, roughly speaking, a restricted form of SGML.

5.0 SUMMARY

In this unit, we considered the extensible markup language (XML). We equally specified the common concepts of XML. Later on, we identified the relationship and differences between: XML and HTML, XML and SGML. You may now proceed to the tutor marked assignment below.

6.0 TUTOR-MARKED ASSIGNMENT

1. What is an extensible markup language?
2. How are the semantics of an XML defined?
3. State the key difference between XML and HTML
4. What is the relationship between XML and SGML?

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D).*Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970)."A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

"Database Design Basics". (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). "Is SQL a Standard Anymore?" *TechRepublic's Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching* .Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al.* (2009). *Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 2 SIGNIFICANCE OF XML

CONTENTS

1.0 INTRODUCTION

The previous unit introduced some general concepts of the extensible markup language (XML). In this unit, we will learn the relevance of XML and its development goals. In order to understand the XML specifications, we shall equally discuss the extensible pointer language, extensible linking language and the extensible style language.

2.0 OBJECTIVES

At the end this unit, you should be able to:

- give a concise description of the import of XML
- identify the development goals of extensible markup languages (XML)
- know how you would view an XML document, assuming you do not have an XML browser
- state the relationship between the XML pointer language and the XML linking language.

3.0 MAIN CONTENT

3.1 Why XML?

In order to recognise the value of XML, it is important to understand why it was created. XML was created so that richly structured documents could be used over the web. This is because other viable alternatives, HTML and SGML, are not practical for this purpose.

HTML, as we have already discussed, comes bound with a set of semantics and does not provide arbitrary structure.

SGML provides arbitrary structure, but is too difficult to implement for a web browser. Full SGML systems solve large, complex problems that justify their cost. Viewing structured documents sent over the web rarely carries such justification.

This is not to say that XML can be expected to completely replace SGML. While XML is being designed to deliver structured content over the web, some of the very features it lacks to make this practical, make SGML a more satisfactory solution for the creation and long-time storage of complex documents. In many organisations, filtering SGML to XML will be the standard procedure for web delivery.

3.2 XML Development Goals

Based on the W3C Recommendation, the Extensible Markup Language (XML) 1.0, XML specification sets out the following goals for XML:

- It shall be straightforward to use XML over the Internet. Users must be able to view XML documents as quickly and easily as HTML documents. In practice, this will only be possible when XML browsers are as robust and widely available as HTML browsers, but the principle remains.
- XML shall support a wide variety of applications. XML should be beneficial to a wide variety of diverse applications: authoring, browsing, content analysis, etc. Although the initial focus is on serving structured documents over the web, it is not meant to narrowly define XML.
- XML shall be compatible with SGML. Most of the people involved in the XML effort come from organisations that have a large, in some cases staggering, amount of material in SGML. XML was designed pragmatically, to be compatible with existing standards while solving the relatively new problem of sending richly structured documents over the web.

- It shall be easy to write programs that process XML documents. The colloquial way of expressing this goal while the spec was being developed is that it ought to take about two weeks for a competent computer science graduate student to build a program that can process XML documents.
- The number of optional features in XML is to be kept to an absolute minimum, ideally zero. Optional features inevitably raise compatibility problems when users want to share documents that sometimes lead to confusion and frustration.
- XML documents should be human-legible and reasonably clear. If you do not have an XML browser and you have received a chunk of XML from somewhere, you ought to be able to look at it in your favorite text editor and actually figure out what the content means.
- The XML design should be prepared quickly. Standards efforts are notoriously slow. XML was needed immediately and developed as quickly as possible.
- The design of XML shall be formal and concise. In many ways a corollary to rule 4, it essentially means that XML must be expressed in EBNF and must be amenable to modern compiler tools and techniques.
  There are a number of technical reasons why the SGML grammar *cannot* be expressed in EBNF. Writing a proper SGML parser requires handling a variety of rarely used and difficult to parse language features. XML does not.
- XML documents should be easy to create. Although, there will eventually be sophisticated editors to create and edit XML content, they may not appear immediately. In the interim, it must be possible to create XML documents in other ways: directly in a text editor, with simple shell and Perl scripts, etc.
- Terseness in XML markup is of minimal importance. Several SGML language features were designed to minimise the amount of typing required to manual keying in SGML documents. These features are not supported in XML. From an abstract point of view, these documents are indistinguishable from their more fully specified forms, but supporting these features adds a considerable burden to the SGML parser (or the person writing it, anyway). In addition, most modern editors offer better facilities to define shortcuts when entering text.

SELF-ASSESSMENT EXERCISE

State why it is desirable to have the number of optional features in XML is to be kept an absolute minimum?

3.3 Defining XML

XML is defined by a number of related specifications:

3.3.1 Extensible Markup Language (XML) 1.0

This defines the syntax of XML. The XML specification is the primary focus of this unit.

3.3.2 XML Pointer Language (XPointer) and XML Linking Language (XLink)

This defines a standard way to represent links between resources. In addition to simple links, like HTML's <A> tag, XML has mechanisms for links between multiple resources and links between read-only resources. XPointer describes how to address a resource, XLink describes how to associate two or more resources.

3.3.3 Extensible Style Language (XSL)

The Extensible Style Language defines the standard style sheet language for XML.

Currently, namespaces (dealing with tags from multiple tag sets), a query language (finding out what is in a document or a collection of documents), and a schema language (describing the relationships between tags, DTDs in XML) are all being actively pursued.

3.3.4 Understanding the Specs

For the most part, reading and understanding the XML specifications does not require extensive knowledge of SGML or any of the related technologies.

4.0 CONCLUSION

To wrap up, in this unit we learnt that: XML was created so that richly structured documents could be used over the web. This is because other viable alternatives, HTML and SGML, are not practical for this purpose. Basically, HTML comes bound with a set of semantics and does not provide arbitrary structure. SGML provides arbitrary structure, but is too difficult to implement for a web browser.

Based on the W3C Recommendation, the Extensible Markup Language (XML) 1.0, XML specification sets out a number of goals for XML.

Some of which are as follows:

- It shall be straightforward to use XML over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs that process XML documents.
- The number of optional features in XML is to be kept to an absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly. Standards efforts are notoriously slow. XML was needed immediately and was developed as quickly as possible.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.

The Extensible Markup Language (XML) 1.0 defines the syntax of XML. XML Pointer Language (XPointer) and XML Linking Language (XLink) define a standard way to represent links between resources. In addition to simple links, like HTML's `<A>` tag, XML has mechanisms for links between multiple resources and links between read-only resources. XPointer describes how to address a resource; XLink describes how to associate two or more resources.

The Extensible Style Language (XSL) defines the standard style sheet language for XML. For the most part, reading and understanding the XML specifications does not require extensive knowledge of SGML or any of the related technologies.

5.0 SUMMARY

In this unit, we learnt the significance of XML and its development goals. We equally considered the extensible pointer language, extensible linking language and the extensible style language. You can now attempt the tutor-marked assignment below. Good luck!

6.0 TUTOR-MARKED ASSIGNMENT

1. Give a concise description of the significance of XML.
2. State the development goals of extensible markup languages (XML).
3. Assuming you do not have an XML browser, how else would you view an XML document?
4. Identify the relationship between the XML pointer language and the XML linking language.

7.0 REFERENCES/FURTHER READING

Avi, S. *et al*. (N.D).*Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?”. *TechRepublic’s Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL* (1st Ed.). Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More.* Morgan Kaufmann Press.

Kawash, J. (2004). “Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus”. *Journal of Computers in Mathematics and Science Teaching*. Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). *Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T. *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 3 XML DOCUMENTS

CONTENTS

1.0 INTRODUCTION

In this unit, you shall learn about the simple XML document as well as XML Markup and Content. You will equally learn about six kinds of markup that can occur in an XML document: elements, entity references, comments, processing instructions, marked sections, and document type declarations.

2.0 OBJECTIVES

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

- describe a simple XML document
- outline the main ideas of a model XML document
- explain the term ‘character references’
- discuss the main role of a CDATA section in a document
- mention the core components of XML documents
- identify six kinds of markup that can occur in an XML document.

3.0 MAIN CONTENT

3.1 A Simple XML Document

Essentially, XML documents appear similar to HTML or SGML. A simple XML document is presented in the next section.

3.2 Typical Model of an XML Document

<?xml version="1.0"?>

<oldjoke>

<burns>Say <quote>goodnight</quote>,

Gracie.</burns>

<allen><quote>Goodnight,

Gracie.</quote></allen>

<applause/>

</oldjoke>

Key Points to note:

The document begins with a processing instruction: **<?xml ...?>**. This is the ***XML declaration***. While it is not required, its presence explicitly identifies the document as an XML document and indicates the version of XML to which it was authored.

There is no document type declaration. Unlike SGML, XML does not require a document type declaration. However, a document type declaration can be supplied, and some documents will require one in order to be understood unambiguously.

Empty elements (<applause/> in this example) have a modified syntax. While most elements in a document are wrappers around some content, empty elements are simply markers where something occurs (a horizontal rule for HTML's <hr> tag, for example or a cross reference for DocBook's <xref> tag). The trailing /> in the modified syntax indicates to a program processing the XML document that the element is empty and no matching end-tag should be sought. Since XML documents do not require a document type declaration, without this clue it could be impossible for an XML parser to determine which tags were

intentionally empty and which had been left empty by mistake. XML has softened the distinction between elements which are declared as EMPTY and elements which merely have no content. In XML, it is legal to use the empty-element tag syntax in either case. It's also legal to use a start-tag/end-tag pair for empty elements: <applause></applause>. If interoperability is of any concern, it is best to reserve empty-element tag syntax for elements which are declared as EMPTY and to only use the empty-element tag form for those elements.

SELF-ASSESSMENT EXERCISE

Describe the concept of 'empty elements' with respect to an XML document.

3.3 XML Markup and Content

XML documents are composed of markup and content. There are six kinds of markup that can occur in an XML document: elements, entity references, comments, processing instructions, marked sections, and document type declarations. The following sections introduce each of these markup concepts. We will take a closer look at document type declarations in the sections that follow.

3.3.1 Elements

Elements are the most common form of markup. Delimited by angle brackets, most elements identify the nature of the content they surround. Some elements may be empty, as seen above, in which case they have no content. If an element is not empty, it begins with a start-tag, <element>, and ends with an end-tag, </element>.

3.3.2 Attributes

Attributes are name-value pairs that occur inside start-tags after the element name. For example,

is a div element with the attribute class having the value preface. In XML, all attribute values must be quoted.

3.3.3 Entity References

In order to introduce markup into a document, some characters have been reserved to identify the start of markup. The left angle bracket, < , for instance, identifies the beginning of an element start- or end-tag. In

order to insert these characters into your document as content, there must be an alternative way to represent them. In XML, entities are used to represent these special characters. Entities are also used to refer to often repeated or varying text and to include the content of external files. Every entity must have a unique name. Defining your own entity names is discussed in the section on entity declarations. In order to use an entity, you simply reference it by name. Entity references begin with the ampersand and end with a semicolon.

For example, the lt entity inserts a literal < into a document. So the string <element> can be represented in an XML document as <element>.

A special form of entity reference, called a ***character reference***, can be used to insert arbitrary Unicode characters into your document. This is a mechanism for inserting characters that cannot be typed directly on your keyboard.

Character references take one of two forms: decimal references, ℞, and hexadecimal references, ℞. Both of these refer to character number U+211E from Unicode (which is the standard Rx prescription symbol, in case you were wondering).

3.3.4 Comments

Comments begin with <!-- and end with -->. Comments can contain any data except the literal string --. You can place comments between markups anywhere in your document.

Comments are not part of the textual content of an XML document. An XML processor is not required to pass them along to an application.

3.3.5 Processing Instruction

Processing instructions (PIs) are an escape hatch to provide information to an application. Like comments, they are not textually part of the XML document, but the XML processor is required to pass them to an application.

Processing instructions have the form: <?name pidata?>. The name, called the PI target, identifies the PI to the application. Applications should process only the targets they recognise and ignore all other PIs. Any data that follows the PI target is optional; it is for the application that recognises the target. The names used in PIs may be declared as notations in order to formally identify them.

PI names beginning with xml are reserved for XML standardisation.

3.3.6 CDATA Sections

In a document, a CDATA section instructs the parser to ignore most markup characters.

Consider a source code listing in an XML document. It might contain characters that the XML parser would ordinarily recognize as markup (< and &, for example). In order to prevent this, a CDATA section can be used.

<![CDATA[

*p = &q;

b = (i <= 3);

]]>

Between the start of the section, <![CDATA[and the end of the section,]]>, all character data is passed directly to the application, without interpretation. Elements, entity references, comments, and processing instructions are all unrecognised and the characters that comprise them are passed literally to the application.

The only string that cannot occur in a CDATA section is]]>.

3.3.7 Document Type Declaration

A large percentage of the XML specification deals with various sorts of declarations that are allowed in XML. If you have experience with SGML, you will recognise these declarations from SGML DTDs (Document Type Definitions). If you have never seen them before, their significance may not be immediately obvious.

4.0 CONCLUSION

In this unit, we were made to understand that XML documents appear similar to HTML or SGML. We equally noted that in an XML

Document:

- The document begins with a processing instruction: <?xml ...?>.
- There’s no document type declaration.

- Empty elements (<applause/> in this example) have a modified syntax. The trailing /> in the modified syntax indicates to a program processing the XML document that the element is empty and no matching end-tag should be sought.

Finally, we discovered that XML documents are composed of markups and content and that there are six kinds of markup that can occur in an XML document: elements, entity references, comments, processing instructions, marked sections, and document type declarations.

5.0 SUMMARY

This unit presented the simple XML document as well as XML Markup and Content. We equally considered six kinds of markup that can occur in an XML document: elements, entity references, comments, processing instructions, marked sections, and document type declarations. In order to assess your comprehension of the just concluded unit, you need to try out the questions that ensue.

6.0 TUTOR-MARKED ASSIGNMENT

1. Give a concise description of the term 'character references'.
2. What is the main role of a CDATA section in a document?
3. State the core components of XML documents.
4. List the six kinds of markup that can occur in an XML document.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D).*Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970)."A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM archive*. Vol.13. Issue 6.pp. 377-387.

"Database Design Basics". (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). "Is SQL a Standard Anymore?" *TechRepublic's Builder.com*. TechRepublic. http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL* (1st Ed.). Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching.* Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM*, 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009).*Database Systems: A Practical Approach to Design, Implementation and Management*.

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 4 DOCUMENT TYPE DECLARATION

CONTENTS

1.0 INTRODUCTION

This unit presents the notion of document type declarations with respect to XML documents. Four main types of XML declarations are identified as follows: Element Type Declarations, Attribute List Declarations, Entity Declarations and Notation Declarations. Do take note of these main ideas. Enjoy your studies.

2.0 OBJECTIVES

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

- explain the notion of document declaration
- identify the components of meta-information
- describe the four main types of declarations in XML
- list and describe the common attribute types.

3.0 MAIN CONTENT

3.1 XML Document Declarations

One of the greatest strengths of XML is that it enables one to create personal tag names. However, for any given application, it is probably not meaningful for tags to occur in a completely arbitrary order. Consider the example given below:
<gracie><quote><oldjoke>Goodnight,

<applause/>Gracie</oldjoke></quote>

<burns><gracie>Say <quote>goodnight</quote>,

</gracie>Gracie.</burns></gracie>

It is so far outside the bounds of what we normally expect that it appears nonsensical. It just does not *mean* anything.

However, from a strictly syntactic point of view, there is nothing wrong with that XML document. So, if the document is to have meaning, and certainly if you are writing a stylesheet or application to process it, there must be some constraint on the sequence and nesting of tags. Declarations are where these constraints can be expressed.

Generally, declarations allow a document to communicate meta-information to the parser about its content. Meta-information includes the allowed sequence and nesting of tags, attribute values and their types and defaults, the names of external files that may be referenced and whether or not they contain XML, the formats of some external (non-XML) data that may be referenced, and the entities that may be encountered.

SELF-ASSESSMENT EXERCISE

What is the basic requirement for writing an application to process an XML document?

3.2 Types of Declaration in XML

There are four kinds of declarations in XML: element type declarations, attribute list declarations, entity declarations, and notation declarations.

3.2.1 Element Type Declarations

Element type declarations identify the names of elements and the nature of their content. A typical element type declaration looks like this:

<!ELEMENT oldjoke (burns+, allen, applause?)>

This declaration identifies the element named oldjoke. Its *content model* follows the element name. The content model defines what an element may contain. In this case, an oldjoke must contain burns and allen and may contain applause. The commas between element names indicate that they must occur in succession. The plus after burns indicates that it may be repeated more than once but must occur at least once. The

question mark after applause indicates that it is optional (it may be absent, or it may occur exactly once). A name with no punctuation, such as allen, must occur exactly once.

Declarations for burns, allen, applause and all other elements used in any content model must also be present for an XML processor to check the validity of a document.

In addition to element names, the special symbol #PCDATA is reserved to indicate character data. The moniker PCDATA stands for parseable character data.

Elements that contain only other elements are said to have *element content*. Elements that contain both other elements and #PCDATA are said to have *mixed content*.

For example, the definition for burns might be

<!ELEMENT burns (#PCDATA | quote)*>

The vertical bar indicates an or relationship, the asterisk indicates that the content is optional (may occur zero or more times); therefore, by this definition, burns may contain zero or more characters and quote tags, mixed in any order. All mixed content models must have this form: #PCDATA must come first, all of the elements must be separated by vertical bars, and the entire group must be optional.

Two other content models are possible: EMPTY indicates that the element has no content (and consequently no end-tag), and ANY indicates that *any* content is allowed. The ANY content model is sometimes useful during document conversion, but should be avoided at almost any cost in a production environment because it disables all content checking in that element.

Here is a complete set of element declarations for Example 1:

Example 2. Element Declarations for Old Jokes

<!ELEMENT oldjoke (burns+, allen, applause?)>

<!ELEMENT burns (#PCDATA | quote)*>

<!ELEMENT allen (#PCDATA | quote)*>

<!ELEMENT quote (#PCDATA)*>

<!ELEMENT applause EMPTY>

3.2.2 Attribute List Declarations

Attribute list declarations identify which elements may have attributes, what attributes they may have, what values the attributes may hold, and what value is the default. A typical attribute list declaration looks like this:

<!ATTLIST oldjoke

name
ID
#REQUIRED

label
CDATA
#IMPLIED

status (funny | notfunny) 'funny'>

In this example, the oldjoke element has ***three attributes***: **name,** which is an ID and is required; **label**, which is a string (character data) and is not required; and **status**, which must be either funny or notfunny and defaults to funny, if no value is specified.

Each attribute in a declaration has **three parts**: a name, a type, and a default value.

You are free to select any name you wish, subject to some slight restrictions, but names cannot be repeated on the same element.

Typical Attribute Types

There are six possible attribute types:

a. CDATA

- CDATA attributes are strings, any text is allowed. Don't confuse CDATA attributes with CDATA sections, they are unrelated.

b. ID

- The value of an ID attribute must be a name [Section 2.3, production 5]. All of the ID values used in a document must be different. IDs uniquely identify individual elements in a document. Elements can have only a single ID attribute.

c. IDREF or IDREFS

- An IDREF attribute value must be that of a single ID attribute on some element in the document. The value of an IDREFS attribute may contain multiple IDREF values separated by white space.

d. ENTITY OR ENTITIES

- An ENTITY attribute value must be the name of a single entity (see the discussion of entity declarations below). The value of an ENTITIES attribute may contain multiple entity names separated by white space.

e. NMTOKEN or NMTOKENS

- Name token attributes are a restricted form of string attribute. In general, an NMTOKEN attribute must consist of a single word, but there are no additional constraints on the word, it does not have to match another attribute or declaration. The value of an NMTOKENS attribute may contain multiple NMTOKEN values separated by white space.

f. A LIST OF NAMES

- One can specify that the value of an attribute must be taken from a specific list of names. This is frequently called an enumerated type because each of the possible values is explicitly enumerated in the declaration. Alternatively, you one can actually specify that the names must match a notation name.

3.2.3 Entity Declarations

Entity declarations enable one associate a name with some other fragment of content. That construct can be a chunk of regular text, a chunk of the document type declaration, or a reference to an external file containing either text or binary data.

A few typical entity declarations are depicted in the example below:

3.2.4 Typical Entity Declarations

```
<!ENTITY
ATI
“ArborText, Inc.”>

<!ENTITY boilerplate      SYSTEM
```

“/standard/legalnotice.xml”>

<!ENTITY ATIlogo
SYSTEM “/standard/logo.gif" NDATA GIF87A>

3.2.5 Types of Entities

There are three kinds of entities:

Internal Entities

Internal entities associate a name with a string of literal text. The first entity in the example above is an internal entity. Using &ATI; anywhere in the document will insert ArborText, Inc. at that location. Internal entities allow you to define shortcuts for frequently typed text or text that is expected to change, such as the revision status of a document.

Internal entities can include references to other internal entities, but it is an error for them to be recursive.

The XML specification predefines five internal entities:

- < produces the left angle bracket, <
- > produces the right angle bracket, >
- & produces the ampersand, &
- ' produces a single quote character (an apostrophe), '
- " produces a double quote character, "

External Entities

External entities associate a name with the content of another file. External entities allow an XML document to refer to the contents of another file. External entities contain either text or binary data. If they contain text, the content of the external file is inserted at the point of reference and parsed as part of the referral document. Binary data is not parsed and may only be referenced in an attribute. Binary data is used to reference figures and other non-XML content in the document.

The second and third entities in section 3.Ent are external entities.

Using &boilerplate; will have insert the *contents* of the file /standard/legalnotice.xml at the location of the entity reference. The XML processor will parse the content of that file as if it occurred literally at that location.

The entity ATIlogo is also an external entity, but its content is binary. The ATIlogo entity can only be used as the value of an ENTITY (or ENTITIES) attribute (on a graphic element, perhaps). The XML processor will pass this information along to an application, but it does not attempt to process the content of /standard/logo.gif.

Parameter Entities

Parameter entities can only occur in the document type declaration. A parameter entity declaration is identified by placing % (percent-space) in front of its name in the declaration. The percent sign is also used in references to parameter entities, instead of the ampersand. Parameter entity references are immediately expanded in the document type declaration and their replacement text is part of the declaration, whereas normal entity references are not expanded. Parameter entities are not recognised in the body of a document.

Looking back at the element declarations in Ex 2, you will notice that two of them have the same content model:

<!ELEMENT burns (#PCDATA | quote)*>

<!ELEMENT allen (#PCDATA | quote)*>

At the moment, these two elements are the same only because they happen to have the same literal definition. In order to make more explicit the fact that these two elements are semantically the same, use a parameter entity to define their content model. The advantage of using a parameter entity is two-fold. First, it allows you to give a descriptive name to the content, and second it allows you to change the content model in only a single place, if you wish to update the element declarations, assuring that they always stay the same:

<!ENTITY % personcontent "#PCDATA | quote">

<!ELEMENT burns (%personcontent;)*>

<!ELEMENT allen (%personcontent;)*>

3.2.6 Notation Declarations

Notation declarations identify specific types of external binary data. This information is passed to the processing application, which may make whatever use of it wishes. A typical notation declaration is:

<!NOTATION GIF87A SYSTEM "GIF">

Do I need a Document Type Declaration?

As we have seen, XML content can be processed without a document type declaration. However, there are some instances where the declaration is required:

Authoring Environments

Most authoring environments need to read and process document type declarations in order to understand and enforce the content models of the document.

Default Attribute Values

If an XML document relies on default attribute values, at least part of the declaration must be processed in order to obtain the correct default values.

White Space Handling

The semantics associated with white space in element content differs from the semantics associated with white space in mixed content. Without a DTD, there is no way for the processor to distinguish between these cases, and all elements are effectively mixed content.

In applications where a person composes or edits the data (as opposed to data that may be generated directly from a database, for example), a DTD is probably going to be required if any structure is to be guaranteed.

Including a Document Type Declaration

If present, the document type declaration must be the first thing in the document after optional processing instructions and comments.

The document type declaration identifies the root element of the document and may contain additional declarations. All XML documents must have a single root element that contains all of the content of the document. Additional declarations may come from an external DTD, called the external subset, or be included directly in the document, the internal subset, or both:

<?XML version="1.0" standalone="no"?>

<!DOCTYPE chapter SYSTEM "dbook.dtd" [

```
<!ENTITY %ulink.module "IGNORE">

<!ELEMENT ulink (#PCDATA)*>

<!ATTLIST ulink

   xml:link       CDATA  #FIXED "SIMPLE"

   xml-attributes CDATA  #FIXED "HREF URL"

   URL            CDATA  #REQUIRED>

]>

<chapter>...</chapter>
```

This example references an external DTD, dbook.dtd, and includes element and attribute declarations for the ulink element in the internal subset. In this case, ulink is being given the semantics of a simple link from the XLink specification.

Note that declarations in the internal subset override declarations in the external subset. The XML processor reads the internal subset before the external subset and the *first* declaration takes precedence.

In order to determine if a document is valid, the XML processor must read the entire document type declaration (both internal and external subsets). But for some applications, validity may not be required, and it may be sufficient for the processor to read only the internal subset. In the example above, if validity is unimportant and the only reason to read the doctype declaration is to identify the semantics of ulink, reading the external subset is not necessary.

4.0 CONCLUSION

In this unit, we were made to understand that XML enables one to create personal tag names. We learnt that in order to ensure that documents are meaningful, there must be some constraint on the sequence and nesting of tags. Thus, constraints can be expressed by means of declarations.

Generally, declarations allow a document to communicate meta-information to the parser about its content. Meta-information includes the allowed sequence and nesting of tags, attributes, values and their types and defaults, the names of external files that may be referenced and whether or not they contain XML, the formats of some external

(non-XML) data that may be referenced, and the entities that may be encountered.

The four kinds of declarations in XML identified are as follows: element type declarations attribute list declarations, entity declarations, and notation declarations.

Element type declarations identify the names of elements and the nature of their content. Attribute list declarations identify which elements may have attributes, what attributes they may have, what values the attributes may hold, and what value is the default.

Six possible attribute types were specified as follows: CDATA, ID, IDREF or IDREFS, Entity or Entities, NMTOKEN or NMTOKENS and A List of Names

Entity declarations enable one associate a name with some other fragment of content. We discovered three kinds of entities: Internal entities, External entities and Parameter entities.

Notation declarations identify specific types of external binary data. This information is passed to the processing application, which may make whatever use of it as it wishes.

In sum, the document type declaration identifies the root element of the document and may contain additional declarations. All XML documents must have a single root element that contains all of the content of the document. Additional declarations may come from an external DTD, called the external subset, or be included directly in the document, the internal subset, or both

Note that declarations in the internal subset override declarations in the external subset. The XML processor reads the internal subset before the external subset and the *first* declaration takes precedence.

We equally learnt that in order to determine if a document is valid, the XML processor must read the entire document type declaration (both internal and external subsets). But for some applications, validity may not be required, and it may be sufficient for the processor to read only the internal subset. In the example above, if validity is unimportant and the only reason to read the doctype declaration is to identify the semantics of ulink, reading the external subset is not necessary.

5.0 SUMMARY

This unit presented details of the concept of document type declarations with respect to XML documents. Four main types of XML declarations were equally specified as follows: Element Type Declarations, Attribute List Declarations, Entity Declarations and Notation Declarations. We hope you enjoyed your studies. It is time to test your knowledge on this subject, so do attempt all the tasks listed in the tutor-marked assignment below.

6.0 TUTOR-MARKED ASSIGNMENT

1. Explain the notion of document declaration.
2. List the components of meta-information.
3. Describe the four main kinds of declarations in XML.
4. Mention at least five attributes types.

7.0 REFERENCES/FURTHER READING

Avi, S.; *et al*. (N.D). *Database System Concepts* (6th ed.). McGraw-Hill

Beaulieu, A. & Mary, E. T. (Eds.). *Learning SQL* (2nd ed.). Sebastapol, CA, USA: O'Reilly.

Beynon-Davies, P. (2004). *Database Systems* (3rd ed.). Palgrave: Basingstoke, UK.

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970).”A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM* 13 (6):pp. 377–387.

Codd, E. F. (1970). “A Relational Model of Data for Large Shared Data Banks”. *Communications of the ACM archive*. Vol. 13. Issue 6.pp. 377-387.

“Database Design Basics”. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS; Portland, Oregon, United States. (1986). pp. 472 – 482.

Doll, S. (2002). “Is SQL a Standard Anymore?”. *TechRepublic’s Builder.com*. TechRepublic.

http://articles.techrepublic.com.com/5100-10878_11-1046268.html. Retrieved 2010-01-07.

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press.

itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Lightstone, S.; *et al*. (2007). *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Press.

Kawash, J. (2004). "Complex Quantification in Structured Query Language (SQL): a Tutorial using Relational Calculus". *Journal of Computers in Mathematics and Science Teaching* . Volume 23, Issue 2, 2004 AACE Norfolk, Virginia.

Mike, C. (2011). "Referential Integrity". http://databases.about.com/: About.com. http://databases.about.com/cs/administration/g/refintegrity.htm. Retrieved 2011-03-17.

Oppel, A. (2004). *Databases Demystified*. San Francisco, CA: McGraw-Hill Osborne Media.

Performance Enhancement through Replication in an Object-Oriented DBM. (1986).pp. 325-336.

Ramakrishnan, R. & Gehrke, J. (2000). .*Database Management Systems* (2nd Ed.). McGraw-Hill Higher Education.

Seltzer, M. (2008). "Beyond Relational Databases". *Communications of the ACM,* 51(7), pp. 52-58.

Teorey, T.; *et al*. (2005). *Database Modeling & Design: Logical Design* (4th ed.). Morgan Kaufmann Press.

Teorey, T.J.; *et al*. (2009). *Database Design: Know it All*. Burlington, MA: Morgan Kaufmann Publishers.

Thomas, C.; *et al*. (2009). *Database Systems: A Practical Approach to Design, Implementation and Management.*

Tsitchizris, D. C. & Lochovsky, F.H. (1982). *Data Models.* Englewood-Cliffs: Prentice-Hall.

UNIT 5 INTRODUCTION TO WEB SERVICES

CONTENTS

1.0 INTRODUCTION

This last unit provides an overview of the basic concepts of web services. Having a basic understanding of how Web Services work, will enable you appreciate how Web Services Resource Framework (WSRF) extends Web Services. Even if you think you already know about Web Services, going through this section would enhance your understanding of the topic. Enjoy your studies!

2.0 OBJECTIVES

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

- describe how Web Services are published
- discuss the procedure involved in accessing information from Web Services
- specify the common advantages and disadvantages of Web Services
- list the components of Web Services Architecture
- describe each component of Web Services Architecture.

3.0 MAIN CONTENT

3.1 Web Services Background

For quite a while now, there has been a lot of thrill about "Web Services," and many companies have begun to rely on them for their enterprise applications. So, what exactly are Web Services? To put it quite simply, Web Services are distributed computing technology (like CORBA, RMI, EJB, etc.) which allow us to create client/server applications.

For example, let us suppose a database is kept with up-to-date information about weather in the United States, and that information has to be distributed to everyone in the world. To do so, the weather information could be published through a Web Service that, given a ZIP code, will provide the weather information for that ZIP code.

3.2 Website or Web Services Publishing

Information on a website is intended for users. Conversely, information which is available through a Web Service will always be accessed by software, never directly by a user (in spite of the fact that there might be a user using the software). Although Web Services rely heavily on existing Web technologies (such as HTTP), they have no relation to web browsers and HTML. Thus, while websites are for users, Web Services are for software.

3.3 Accessing Information from Web Services

In order to access information (say weather information) from Web Services, the ***clients*** (i.e. programs that want to access the weather information) would contact the ***Web Service*** (in the *server*), and send a ***service request*** asking for the weather information. The server would then return the forecast through a ***service response***. Obviously, this is an imprecise example of how a Web Service works. Details of how Web Services work is illustrated bel**ow:**

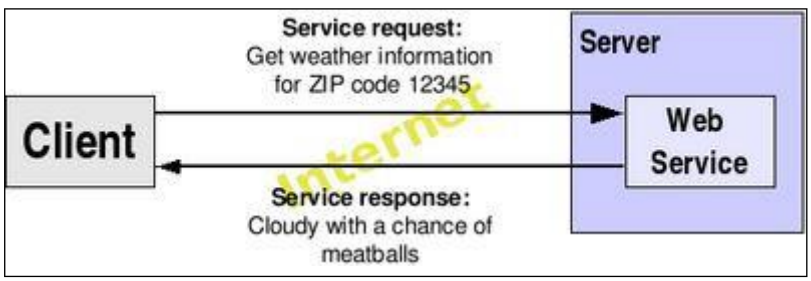

Fig. 5.1 Web Services

3.4 Advantages of Web Services

So, what makes Web Services special? Well, Web Services have certain advantages over other technologies:

a. Web Services are platform-independent and language-independent, since they use standard XML languages. This means that my client program can be programmed in C++ and running under Windows, while the Web Service is programmed in Java and running under Linux.
b. Most Web Services use HTTP for transmitting messages (such as the service request and response). This is a major advantage if you want to build an Internet-scale application, since most of the Internet's proxies and firewalls will not mess with HTTP traffic (unlike CORBA, which usually has trouble with firewalls).

3.5 Disadvantages of Web Services

a. Overhead. Transmitting all your data in XML is obviously not as efficient as using a proprietary binary code. What you win in portability, you lose in efficiency. Even so, this overhead is usually acceptable for most applications, but you will probably never find a critical real-time application that uses Web Services.
b. Lack of versatility. Currently, Web Services are not very versatile, since they only allow for some very basic forms of service invocation. CORBA, for example, offers programmers a lot of supporting services (such as persistency, notifications, lifecycle management, transactions, etc.). Fortunately, there is a lot of emerging Web services specifications (including WSRF) that are helping to make Web services more and more versatile.

However, there is one important characteristic that distinguishes Web Services. While technologies such as CORBA and EJB are geared towards *highly coupled* distributed systems, where the client and the server are very dependent on each other, Web Services are more adequate for *loosely coupled* systems, where the client might have no prior knowledge of the Web Service until it actually invokes it. Highly coupled systems are ideal for intranet applications, but perform poorly on an Internet scale. Web Services, however, are better suited to meet the demands of an Internet-wide application, such as grid-oriented applications.

SELF-ASSESSMENT EXERCISE

State two widespread messages transmitted by Web services using HTTP.

3.6 A Typical Web Service Invocation

In order to understand what a web service invocation entails, we will need to take a look at all the steps involved in a complete Web Service invocation.

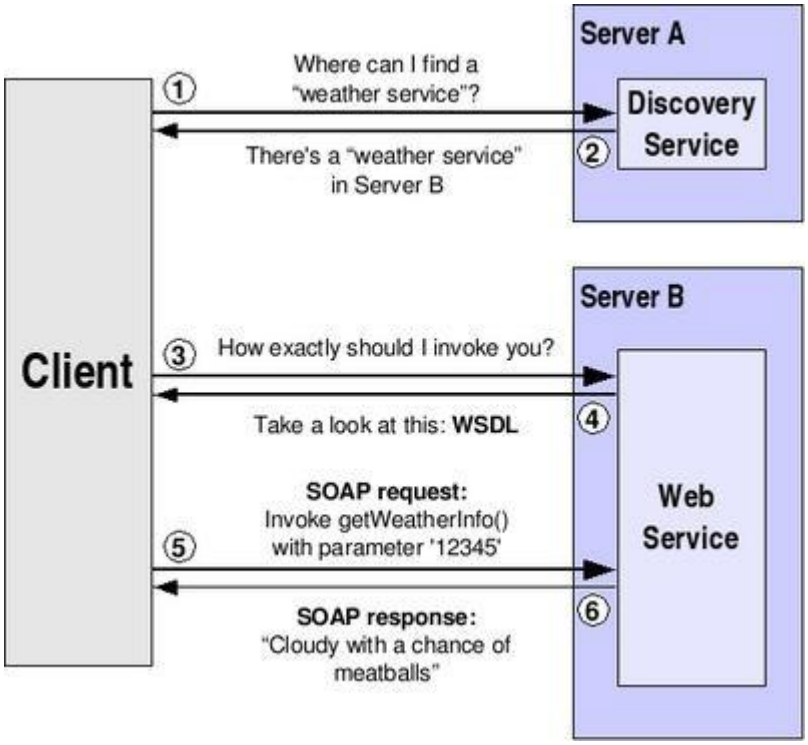

Fig. 5.2: A Typical Web Service Invocation

1. Ordinarily, a client may have no knowledge of what Web Service it is going to invoke. So, our first step will be to ***discover a Web Service that meets our requirements***. For example, we might be interested in locating a public Web Service which can give us the weather forecast in Nigerian cities. We will do this by contacting a *discovery service* (which is itself a Web service).
2. The discovery service will reply, telling us what servers can provide us the service we require.
3. We now know the location of a Web Service, but we have no idea of how to actually invoke it. Sure, we know it can give us the forecast for a Nigerian city, but how do we perform the actual service invocation? The method we have to invoke might be called "**string getCityForecast(int CityPostalCode)**", but it

could also be called "**string getNigerianCityWeather(string cityName, bool isFarenheit)**". We have to ask the Web Service to *describe* itself (i.e. tell us how exactly we should invoke it)

4. The Web Service replies in a language called WSDL.
5. We finally know where the Web Service is located and how to invoke it. The invocation itself is done in a language called SOAP. Therefore, we will first send a *SOAP request* asking for the weather forecast of a certain city.
6. The Web Service will kindly reply with a *SOAP response* which includes the forecast we asked for, or maybe an error message if our SOAP request was incorrect.

3.7 Web Services Architecture

Most of the Web Services Architecture is specified and standardised by the World Wide Web Consortium, the same organisation responsible for XML, HTML, CSS, etc. However, Web Services Architecture essentially consists of:

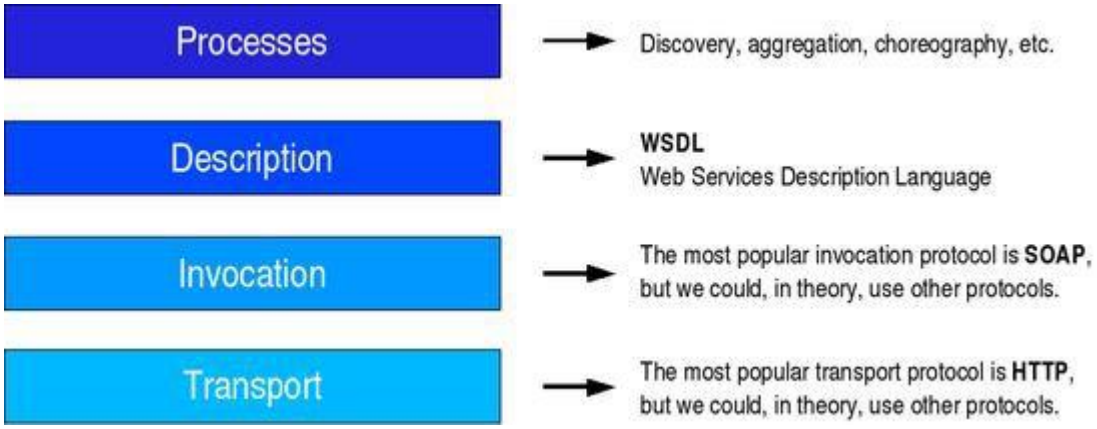

Fig. 5.3: The Web Services Architecture

3.7.1 Service Process

This part of the architecture generally involves more than one Web service. For example, discovery belongs in this part of the architecture, since it allows us to locate one particular service from among a collection of Web services.

3.7.2 Service Description

One of the most interesting features of Web Services is that they are ***self-describing***. This means that, once you have located a Web Service, you can ask it to 'describe itself' and tell you what operations it supports

and how to invoke it. This is handled by the Web Services Description Language (WSDL).

3.7.3 Service Invocation

Invoking a Web Service (and, in general, any kind of distributed service such as a CORBA object or an Enterprise Java Bean) involves passing messages between the client and the server. SOAP (Simple Object Access Protocol) specifies how we should format requests to the server, and how the server should format its responses. In theory, we could use other service invocation languages (such as XML-RPC, or even some *ad hoc* XML language). However, SOAP is by far the most popular choice for Web Services.

3.7.4 Transport

Finally, all these messages must be transmitted somehow between the server and the client. The protocol of choice for this part of the architecture is HTTP (HyperText Transfer Protocol), the same protocol used to access conventional web pages on the Internet. Again, in theory we could be able to use other protocols, but HTTP is currently the most used.

4.0 CONCLUSION

To wrap up, we will go over the main ideas discussed in this unit. Fundamentally, Web Services are distributed computing technology (such as CORBA, RMI, EJB, etc.) which enables one to create client/server applications.

Information which is available through a Web Service will always be accessed by software, never directly by a user (in spite of the fact that there might be a user using the software). Although Web Services rely heavily on existing Web technologies (such as HTTP), they have no relation to web browsers and HTML. Thus, while websites are for users, Web Services are for software.

Web Services have certain advantages over other technologies:

- They are platform-independent and language-independent
- Most Web Services use HTTP for transmitting messages

On the other hand, Web Services have certain disadvantages as follows:

- Critical real-time application cannot be efficiently deployed via Web Services.
- Web Services are not very versatile, since they only allow for some very basic forms of service invocation.

However, there is one important characteristic that distinguishes Web Services. While technologies such as CORBA and EJB are geared towards *highly coupled* distributed systems, where the client and the server are very dependent on each other, Web Services are more adequate for *loosely coupled* systems, where the client might have no prior knowledge of the Web Service until he/she actually invokes it.

Most of the Web Services Architecture is specified and standardised by the World Wide Web Consortium, the same organisation responsible for XML, HTML, CSS, etc. However, Web Services Architecture essentially consists of: Service Processes, Service Descriptions, Service Invocations and Transport.

We hope that the ideas presented in this unit and the previous units have enlightened you on the subjects focusing on database systems, structures their implementations and management, as well basic concepts of XML and Web Services. It is however recommended that you go over the course material and all the references for further reading.

5.0 SUMMARY

This unit presented the background of Web Services as well as Website/ Web Services Publishing. We also learnt how to access information from Web Services. Web Services Invocation and Architecture were equally highlighted. We hope that you found this course interesting and wish you the very best in your studies! However, before you close this page, do attempt the tasks specified in the tutor-marked assignment below.

6.0 TUTOR-MARKED ASSIGNMENT

1. How is Web Services Published?
2. Outline the procedure involved in accessing information from Web Services
3. State at least two advantages of Web Services
4. List the components of Web Services Architecture
5. Give a brief description of Service Invocation

7.0 REFERENCES/FURTHER READING

Beynon-Davies, P. (2004). *Database Systems* (3rd ed). Palgrave: Basingstoke.

Borja, S. (2005). *The Globus Toolkit 4 Programmer's Tutorial.*

Byers, F. R. (2003). Care and Handling of CDs and DVDs — A Guide for Librarians and Archivists. National Institute of Standards and Technology.

Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM archive*. Vol. 13, pp.377-387.

Codd, E. F. (1970)."A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM* 13 (6):pp. 377–387.

Database Design Basics. (N.D.). Retrieved May 1, 2010, from http://office.microsoft.com/en-us/access/HA012242471033.aspx

Development of an Object-Oriented DBMS;(1986). Portland, Oregon, United States; pp. 472 – 482.

Elliote, R.W. & Scott, M.W. (2010). *XML in a Nutshell.*

Gehani, N. (2006). *The Database Book: Principles and Practice using MySQL*. Summit, NJ: Silicon Press

itl.nist.gov (1993). *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

Norman, W. (1998). *A Technical Introduction to XML.*