DAM 344: SEMANTIC DATA MODELLING

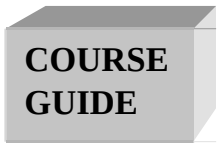
NATIONAL OPEN UNIVERSITY OF NIGERIA

FACULTY OF SCIENCE

COURSE CODE: DAM 344

COURSE TITLE:

SEMANTIC DATA MODELLING



DAM344
SEMANTIC DATA MODELLING

Course Team: Prof. Oludele Awodele (Developer/Writer)-BABCOCK
Greg O. Onwodi (Coordinator) - NOUN
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

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

Published By:
National Open University of Nigeria

First Printed 2012

ISBN: 978-058-851-5

CONTENTS **PAGE**

Course Guide

Introduction

DAM344 - Semantic Data Modelling is a 3- credit unit course. Today, with the increasing diversity and complexity of computer and its application in various fields, the understanding of the concept of Semantic Data Modelling has become inevitably important for individuals and corporate organisations. This is because, the logical data structure of a DBMS(Data Base Management System), whether hierarchical, network, or relational, cannot totally satisfy the requirements for a conceptual definition of data because it is limited in scope and biased toward the implementation strategy employed by the DBMS. Therefore, the need to define data from a conceptual view has led to the development of semantic data modelling techniques.

This course covers principal topics in understanding the general concepts of data modelling, its benefits and limitations. The principles of semantic data modelling, semantic data models and semantic introduction to database, as well as its applications in the computer and business environment, also fall within the content of this course.

It is a course for B.Sc. Computer Science major students, and is normally taken in the third year of the programme duration. It should therefore appeal to whosoever is concerned with the design and implementation of database management system and data modelling generally. To this end, such person should find this material valuable.

This course is divided into three modules. The first module deals with the overview of the data modelling, its concepts and data models. The second module covers, overview of semantic data modelling, its concepts and semantic data models. The third module which is the concluding module of the course discusses the areas of application of semantic data modelling.

This course guide gives you a brief overview of the course contents, course duration, and course materials.

Course Competencies

First, students will learn the key techniques in semantic data modelling specifically, in consonance with the requirement of database system.

Second, students will understand the rationale behind modelling data in a database environment, and improve their modelling skills accordingly.

Last, it will help build the foundation for students to pursue research in the area of semantic data modelling and its application in other fields of life.

Course Objectives

Certain objectives have been set out for the achievement of the course aims. And apart from the course objectives, each unit of this course has its objectives, which you need to confirm if are met, at the end of each unit. So, upon the completion of this course, you should be able to:

- describe what semantic data modelling is and why it is required in the database management system and database environment
- explain the major concepts and techniques involved in the task of general data modelling, and semantic data modeling
- write semantic data language and use such to the advantage of organisation's database management system, in the database environment
- build, operate, and maintain systems and interfaces, in a cost effective manner, bearing in mind how different applications can share data**.**

Working through this Course

In order to have a thorough understanding of the course units, you will need to read and understand the contents, practice the steps and techniques involved in the task of semantic data modelling, and be committed to its application in databases.

This course is designed to cover approximately seventeen weeks, and requires your devoted attention, answering the exercises in the tutor- marked assignments and gets them submitted to your tutors.

Study Units

There are 8 units in this course:

Module 1 **Concepts of Data Modelling**
Unit 1 Overview of Data Modelling
Unit 2 Data Modelling Concepts
Unit 3 Data Models

Module 2 **Semantic Data Modelling**
Unit 1 Overview of Semantic Data Modelling
Unit 2 Semantic Data Models
Unit 3 Semantic Data Modelling Concepts

Module 3 **Areas of Application of Semantic Data Modelling**
Unit 1 Application in Computer
Unit 2 Application in Business

You should make use of the course materials, and do the exercises to enhance your learning.

References and Further Readings

BLAHA, M. (2017). *PATTERNS OF DATA MODELING*. Place of publication not identified: CRC Press.

Connolly, T. M., & Begg, C. E. (2015). *Database systems: A practical approach to design, implementation, and management*. Upper Saddle River, NJ: Pearson.

Hoberman, S. (2013). *Data modeling made simple: With Embarcadero ER/Studio data architect*. Westfield, NJ: Technic Publications.

Hogan, R. (2020). *A practical guide to database design*. Boca Raton: Chapman & Hall/CRC.

Logical Database Design: Relational Modeling and Normalization. (2014). *Developing Windows-Based and Web-Enabled Information Systems*, 108-137. doi:10.1201/b16616-12
, T. J. (2011).

Database modeling and design: Logical design. Amsterdam: Elsevier.

Hammer, M. & McLeod, D. Database Description with SDM: *A Semantic Database Model.*
Jack, C. & Colin, K. (2006). "Semantic Data Models." Unpublished Essay. School of Computing, Napier University. 15 Oct. 2006.

Hills, T. (2019). *NoSQL and SQL data modeling: Bringing together data, semantics, and software*. New York: Technics publications.

Hussein, T., Paulheim, H., Lukosch, S., Ziegler, J., & Calvary, G. (2016). *Semantic models for adaptive interactive systems*. London: Springer.

Johan, T. B. (1992). *Semantic Data Modelling*. Prentice Hall.

Len, S. & Paul, A. (2008). The Data Model Resource Book: *Universal Patterns for Data Modelling* Volume 3. John Wiley & Sons.

Len, S.; Inmon, W.H. and Kent, G. (2007). *The Data Model Resource Book*. Wiley, 1997. ISBN 0-471-15364-8. Reviewed by Van Scott on tdan.com. Accessed 1 Nov 2008.

Martin, E. M. (1992). *Data Analysis, Data Modelling, and Classification*.

Matthew, W. & Julian, F. (1999). *Developing High Quality Data Models*.

The European Process Industries STEP Technical Liaison Executive (EPISTLE).

Peckham, J. & Maryanski, F. Semantic Data Models. *ACM Comput. Surv.* 20, 3 (Sept.1988), pp.153−189.

Potter, W. D. & Trueblood, R. P. *Traditional, Semantic, and Hyper- semantic Approaches to Data Modelling*. Computer 21, 6 (June 1988), pp. 53−63.

Qin, Z., & Tang, Y. (2014). *Uncertainty modeling for data mining a label semantics approach*. Hangzhou: Zhejiang Univ. Press.

Relational Data Model in DBMS: Concepts, Constraints, Example. (n.d.). Retrieved from https://www.guru99.com/relational-data-model-dbms.html

Relational Model in DBMS. (2019, August 20). Retrieved from https://www.geeksforgeeks.org/relational-model-in-dbms/

Reingruber, M. C. &William, W. G. (1994). The Data Modelling Handbook. *A Best-Practice Approach to Building Quality Data Models*. John Wiley & Sons, Inc.

Semantic Data Model. 13 Aug. 2001. COMTECO Ltd. 15 Oct. 2006 <http://www.comteco.ru/EN/DATAMODE/datamode.htm

Tina, Y. and Lee Y. (1999). "Information Modelling from Design to Implementation" National Institute of Standards and Technology

Tutcher, J. (2015). *Development of semantic data models to support data interoperability in the rail industry*. Birmingham: University of Birmingham.

Uzun, A. (2019). *Semantic Modeling and Enrichment of Mobile and WiFi Network Data*. Cham: Springer International Publishing.

What is Data Modelling? Conceptual, Logical, & Physical Data Models. (n.d.). Retrieved from https://www.guru99.com/data-modelling-conceptual-logical.html

Whitten, J. L.; Lonnie, D. B.; Kevin, C. D. (2004). *Systems Analysis and Design Methods* (6th ed.). ISBN 025619906X.

Presentation Schedule

The Presentation Schedule included in your course materials gives you the important dates for the completion of tutor marked assignments and attending tutorials. Remember, you are required to submit all your assignments by the due date. You should guard against lagging behind in your work.

Assessment

There are two aspects to the assessment of the course. First are the tutor marked assignment; second, is a written examination.

In tackling the assignments, you are expected to apply information and knowledge acquired during this course. The assignments must be submitted to your tutor, for formal assessment in accordance with the deadlines stated in the assignment file.

The work you submit to your tutor for assessment will count for 30% of your total course mark. At the end of the course, you will need to sit for a final three-hour examination. This also accounts for 70% of your total course mark.

How to get the Most from the Course

In distance learning, the study units replace the university lecturer. This is one of the great advantages of distance learning; you can read and work through specially designed study materials, at your own pace, and at a time and place that suit you best. Think of it as reading the lecture instead of listening to a lecturer. In the same way that a lecturer might set you some reading to do, the study units tell you when to read your set books or other material. Just as a lecturer might give you an in-class exercise, your study units provides exercises for you to do at appropriate points.

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 is a set of learning objectives. These objectives enable you know what you should be able to do by the time you have completed the unit. You should use these objectives to guide your study. When you have finished the units, you must go back and check whether you have achieved the objectives, in order to significantly improve your chances of passing the course.

Remember that your tutor's job is to assist you. When you need help, don't hesitate to call and ask your tutor to provide it.

1. Read this course guide thoroughly.
2. 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 assignments relate to the units. Whatever method you choose to use, you should decide on it and write in your own date, for working on each unit.
3. 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 lag behind in their course work.
4. Turn to unit 1 and read the introduction and objectives for the unit.
5. Assemble the study materials. Information about what you need for a unit is given in the 'overview' at the beginning of each unit. You will almost always need both the study unit you are working on and one of your set of books on your desk at the same time.
6. 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 instructed to read sections from your set of books or other articles. Use the unit to guide your reading.
7. Review the objectives for each study unit to confirm that you have achieved them. If you are not sure about any of the objectives, review the study material or consult your tutor.
8. When you are confident that you have achieved a unit's objectives, you can then 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.
9. 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 schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor- marked assignment form and also on the assignment. Consult your tutor as soon as possible, if you have any question or problem.

10. 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

There are 12 hours of tutorials provided in support of this course. You will be notified of dates, times and locations of these tutorials, 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 on any difficulty you might encounter, and provide assistance to you during the course. You must mail or submit your tutor-marked assignments to your tutor well before the due date (at least two working days are required). They will be marked by your tutor and returned to you as soon as possible.

Do not hesitate to contact your tutor by telephone, or e-mail 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 assigned reading
- you have difficulty with the self-test or exercises
- you have a question or problem 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 course tutorials, prepare a question list before attending them. You will learn a lot from participating in discussions actively.

Course Information

Course Code: DAM 344
Course Title: Semantic Data Modelling
Credit Unit: 3 Units
Course Status:
Course Blurb: This course covers principal topics in understanding the general concepts of data modelling, its benefits and limitations. The principles of semantic data modelling, semantic data models and semantic introduction to database, as well as its applications in the computer and business environment, also fall within the content of this course.
Semester:
Course Duration:
Required Hours for Study

Course Team

Course Developer: ACETEL Course Writer:
Content Editor: Prof. Oludele Awodele
Dept. of Computer Science
Babcock University, Ilishan-Remo, Ogun State
Instructional Designer:
Learning Technologists:
Copy Editor

Ice Breaker

Describe yourself in three words
If you could solve one of the world's greatest problems, which one would you choose?
Where do you want to live after you finish school? Why?

Module 1: Concepts of Data Modelling

Module Introduction

This module is divided into three units.

Unit 1: Data Models
Unit 2: Overview of Data Modelling
Unit 3: Data Modelling Concepts

Unit 1: DATA MODELS

Contents

1.0 Introduction

The goal of reading this section is not only for IT professionals to learn how to become a data modeller but also to gain an appreciation of what is involved in data models. And for an information system to be useful, reliable, adaptable, and economic, it must be based first on sound data modelling, and only secondarily on process analysis. So, every IT professional should be prepared to be involved in the creation of such models, be able to read an existing data models, understand when and when not to create a data model.

2.0 Intended Learning Outcomes (ILOs)

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

- explain what data model is all about
- mention and explain the types of data models
- mention the features of a good data model
- mention and explain the steps involved in the task of data modeling
- state the advantages and disadvantages of data models
- state the limitations of data model

3.0 Main Content

3.1 What are Data Models?

This data model is a conceptual representation of

- Data objects
- The associations between different data objects
- The rules.

Data model emphasizes on what data is needed and how it should be organized instead of what operations need to be performed on the data. Data Model is like architect's building plan which helps to build a conceptual model and set the relationship between data items.

Data Models ensure consistency in naming conventions, default values, semantics, security while ensuring quality of the data.

3.2 Why use Data Model?

The primary goals of using data model are:

- Ensures that all data objects required by the database are accurately represented. Omission of data will lead to creation of faulty reports and produce incorrect results.
- A data model helps design the database at the conceptual, physical and logical levels.
- Data Model structure helps to define the relational tables, primary and foreign keys and stored procedures.

- It provides a clear picture of the base data and can be used by database developers to create a physical database.
- It is also helpful to identify missing and redundant data.
- Though the initial creation of data model is labor and time consuming, in the long run, it makes your IT infrastructure upgrade and maintenance cheaper and faster.

3.3 Types of Data Models

There are mainly three different types of data models:

1. **Conceptual:** This Data Model defines **WHAT** the system contains. This model is typically created by Business stakeholders and Data Architects. The purpose is to organize, scope and define business concepts and rules.
2. **Logical:** Defines **HOW** the system should be implemented regardless of the DBMS. This model is typically created by Data Architects and Business Analysts. The purpose is to developed technical map of rules and data structures.
3. **Physical**: This Data Model describes **HOW** the system will be implemented using a specific DBMS system. This model is typically created by DBA and developers. The purpose is actual implementation of the database.

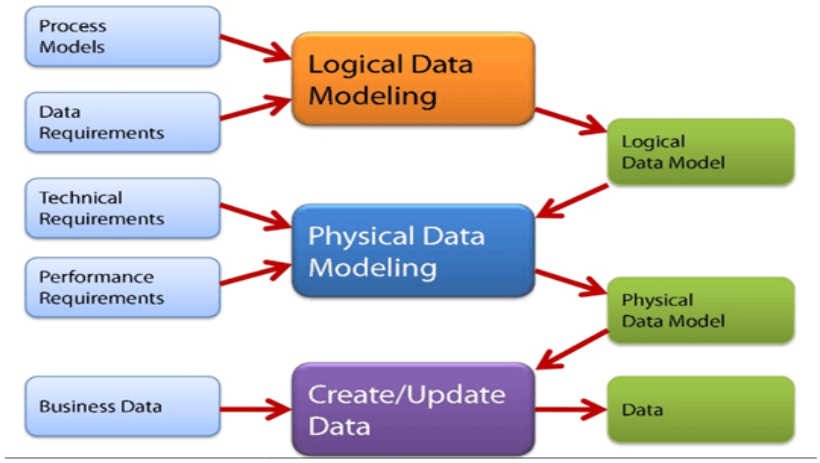


Fig 1: Types of Data Model

3.3.1 Conceptual Data Model

The main aim of this model is to establish the entities, their attributes, and their relationships. In this Data modeling level, there is hardly any detail available of the actual Database structure.

The 3 basic tenants of Data Model are

Entity: A real-world thing

Attribute: Characteristics or properties of an entity

Relationship: Dependency or association between two entities

For example:

- Customer and Product are two entities. Customer number and name are attributes of the Customer entity

- Product name and price are attributes of product entity
- Sale is the relationship between the customer and product

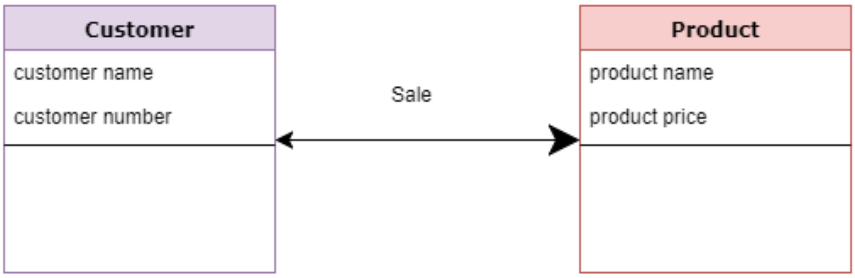


Fig 2: Example of conceptual data model

Characteristics of a conceptual data model

- Offers Organisation-wide coverage of the business concepts.
- This type of Data Models are designed and developed for a business audience.
- The conceptual model is developed independently of hardware specifications like data storage capacity, location or software specifications like DBMS vendor and technology. The focus is to represent data as a user will see it in the "real world."

Conceptual data models known as Domain models create a common vocabulary for all stakeholders by establishing basic concepts and scope.

3.3.2 Logical Data Model

Logical data models add further information to the conceptual model elements. It defines the structure of the data elements and set the relationships between them.

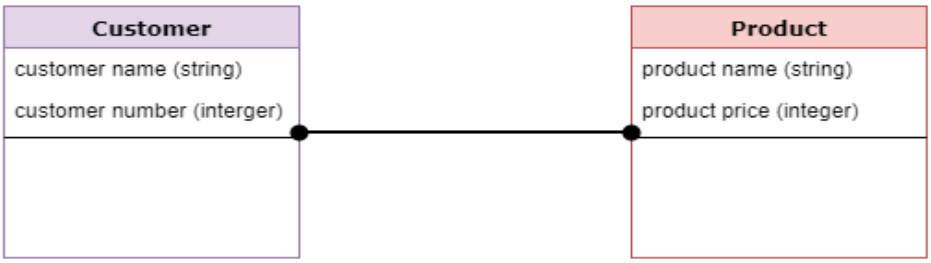


Fig 3: Example of logical data model

The advantage of the Logical data model is to provide a foundation to form the base for the Physical model. However, the modeling structure remains generic.

At this Data Modeling level, no primary or secondary key is defined. At this Data modeling level, you need to verify and adjust the connector details that were set earlier for relationships.

Characteristics of a Logical data model

- Describes data needs for a single project but could integrate with other logical data models based on the scope of the project.
- Designed and developed independently from the DBMS.
- Data attributes will have datatypes with exact precisions and length.
- Normalization processes to the model is applied typically till 3NF.

3.3.3 Physical Data Model

A Physical Data Model describes the database specific implementation of the data model. It offers an abstraction of the database and helps generate schema. This is because of the richness of meta-data offered by a Physical Data Model.

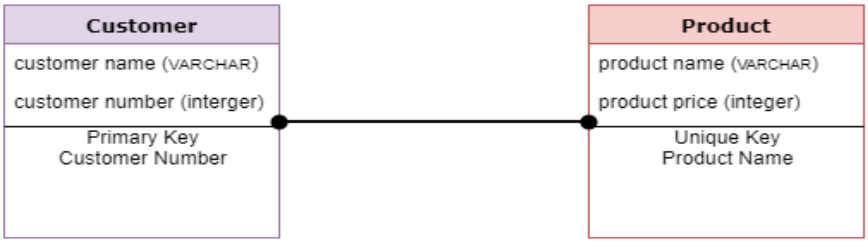


Fig 4: Example of physical data model

This type of Data model also helps to visualize database structure. It helps to model database columns keys, constraints, indexes, triggers, and other RDBMS features.

Characteristics of a physical data model:

- The physical data model describes data need for a single project or application though it may be integrated with other physical data models based on project scope.
- Data Model contains relationships between tables that which addresses cardinality and nullability of the relationships.
- Developed for a specific version of a DBMS, location, data storage or technology to be used in the project.
- Columns should have exact data types, lengths assigned and default values.
- Primary and Foreign keys, views, indexes, access profiles, and authorizations, etc. are defined.

3.4 How to Model Data

It is critical for an application developer to have a grasp of the fundamentals of data modelling so they can not only read data models but also work effectively with Databases who are responsible for the data-oriented aspects of your project.

The following tasks are performed in an iterative manner:

- Identify entity types
- Identify attributes
- Apply naming conventions
- Identify relationships
- Apply data model patterns
- Assign keys
- Normalise to reduce data redundancy
- Denormalise to improve performance

3.4.1 Identify Entity Types

An entity type, also simply called entity (not exactly accurate terminology, but very common in

practice), is similar conceptually to object-orientation's concept of a class – an entity type represents a collection of similar objects. An entity type could represent a collection of people, places, things, events, or concepts. Examples of entities in an order entry system would include Customer, Address, Order, Item, and Tax. Ideally an entity should be normal data modelling world's version of cohesive. A normal entity depicts one concept, just like a cohesive class models one concept. For example, customer and order are clearly two different concepts; therefore it makes sense to model them as separate entities.

3.4.2 Identify Attributes

Each entity type will have one or more data attributes. For example, in Figure 17 you saw that the *Customer* entity has attributes such as *First Name* and *Surname* and in Figure 18 that the *TCUSTOMER* table had corresponding data columns *CUST_FIRST_NAME* and *CUST_SURNAME* (a column, is the implementation of a data attribute within a relational database).

Attributes should also be cohesive from the point of view of your domain, something that is often a judgment call. – in Figure 17 we decided that we wanted to model the fact that people had both first and last names instead of just a name (e.g. "Scott" and "Ambler" vs. "Scott Ambler") whereas we did not distinguish between the sections of zip code (e.g. 90210-1234-5678).

Getting the level of detail right can have a significant impact on your development and maintenance efforts. This is because, over-specifying an attribute (e.g. having three attributes for zip code when you only needed one) can result in overbuilding your system and hence you incur greater development and maintenance costs than you actually needed.

3.4.3 Apply Data Naming Conventions

Your organisation should have standards and guidelines applicable to data modelling, something you should be able to obtain from your enterprise administrators (if they don't exist you should lobby to have some put in place). These guidelines should include naming conventions for both logical and physical modelling; the logical naming conventions should be focused on human readability whereas the physical naming conventions will reflect technical considerations.

The basic idea is that developers should agree to and follow a common set of modelling standards on a software project. Just like there is value in following common coding conventions, clean code that follows your chosen coding guidelines is easier to understand and evolve than code that doesn't, there is similar value in following common modelling conventions.

3.4.4 Identify Relationships

In the real world entities have relationships with other entities. For example, customers PLACE orders, customers LIVE AT addresses, and line items ARE PART OF orders. Place, live at, and are part of, are all terms that define relationships between entities. The relationships between

entities are conceptually identical to the relationships (associations) between objects.
Figure 5 below depicts a partial LDM (Logical Data Model) for an online ordering system. The first thing to notice is the various styles applied to relationship names and roles – different relationships require different approaches. For example the relationship between 'Customer' and 'Order*'* has two names, (places and *is placed by)*, whereas the relationship between 'Customer*'* and 'Address*'* has one. In this example having a second name on the relationship, the idea being that you want to specify how to read the relationship in each direction is redundant – you're better off to find a clear wording for a single relationship name, decreasing the clutter on your diagram. Similarly you will often find that by specifying the roles that an entity plays in a relationship will often negate the need to give the relationship a name (although some CASE tools may inadvertently force you to do this).

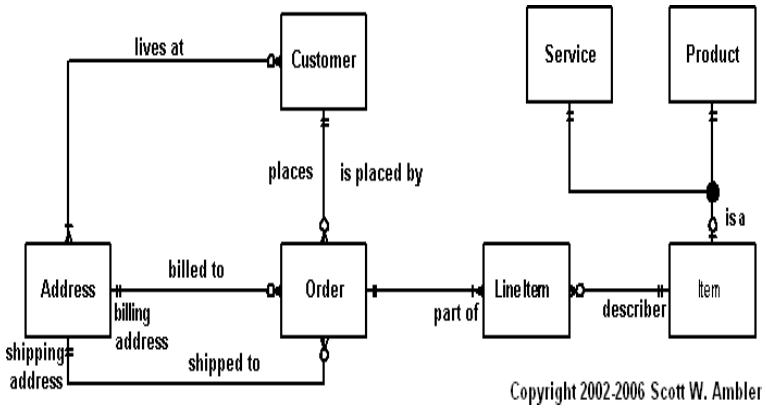


Fig. 5: A Logical Data Model (Information Engineering Notation)

You also need to identify the cardinality and optionality of a relationship. Cardinality represents the concept of "how many" whereas optionality represents the concept of "whether you must have something." For example, it is not enough to know that customers place orders. How many orders can a customer place? None, one, or several? Furthermore, relationships are two-way streets: not only do customers place orders, but orders are placed by customers. This leads to questions like: how many customers can be enrolled in any given order and is it possible to have an order with no customer involved?

3.4.5 Apply Data Model Patterns

Some data modellers will apply common data model patterns, just as object-oriented developers will apply analysis patterns and design patterns. Data model patterns are conceptually closest to analysis patterns because they describe solutions to common domain issues.

3.4.6 Assign Keys

There are two fundamental strategies for assigning keys to tables:
First, you could assign a natural key which is one or more existing data attributes that are unique

to the business concept. In the Customer table of Figure 6 there are two candidate keys, in this case Customer Number and Social Security Number.

Second, you could introduce a new column, called a surrogate key, which is a key that has no business meaning. An example of which is the Address ID column of the Address table in Figure 6. Addresses don't have an "easy" natural key because you would need to use all of the columns of the Address table to form a key for itself (you might be able to get away with just the combination of Street and Zip Code depending on your problem domain), therefore introducing a surrogate key is a much better option in this case.

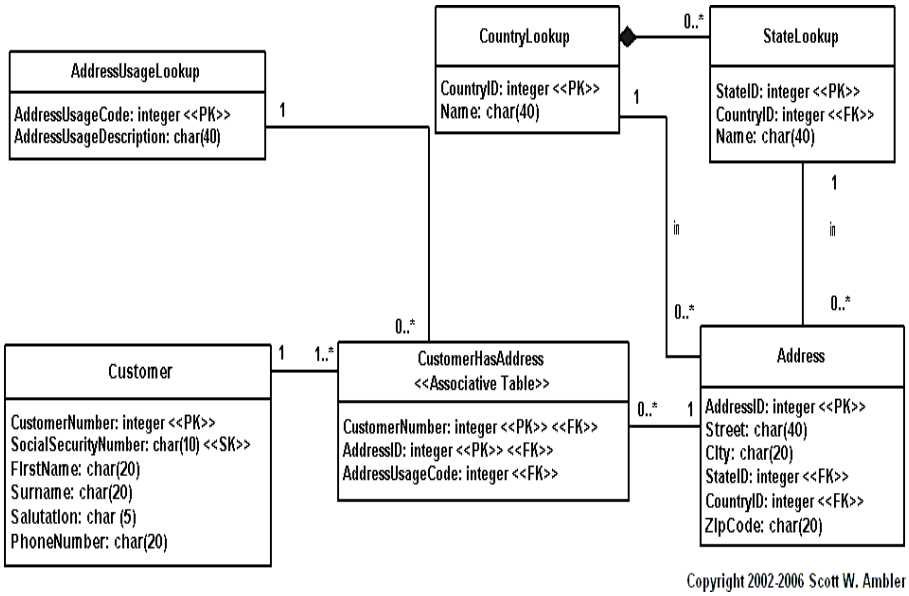


Fig. 6: Customer and Address Revisited (UML Notation)

Let's consider Figure 6 in more detail. Figure 6 presents an alternative design to that presented in Figure 18, a different naming convention was adopted and the model itself is more extensive. In Figure 6 the *Customer* table has the *CustomerNumber* column as its primary key and *SocialSecurityNumber* as an alternate key. This indicates that the preferred way to access customer information is through the value of a person's customer number although your software can get at the same information if it has the person's social security number.

The *CustomerHasAddress* table has a composite primary key, the combination of *CustomerNumber* and *AddressID*. A foreign key is one or more attributes in an entity type that represents a key, either primary or secondary, in another entity type. Foreign keys are used to maintain relationships between rows. For example, the relationships between rows in the *CustomerHasAddress* table and the *Customer* table is maintained by the *CustomerNumber* column within the *CustomerHasAddress* table

The interesting thing about the *CustomerNumber* column is the fact that it is part of the primary key for *CustomerHasAddress* as well as the foreign key to the *Customer* table. Similarly, the *AddressID* column is part of the primary key of *CustomerHasAddress* as well as a foreign key to the *Address* table to maintain the relationship with rows of *Address*.

3.4.7 Normalise to Reduce Data Redundancy

Data normalisation is a process in which data attributes within a data model are organised to increase the cohesion of entity types. In other words, the goal of data normalisation is to reduce and even eliminate data redundancy, an important consideration for application developers because it is incredibly difficult to stores objects in a relational database that maintains the same information in several places.

Table 1 summarises the three most common normalisation rules describing how to put entity types into a series of increasing levels of normalisation. With respect to terminology, a data schema is considered to be at the level of normalisation of its least normalised entity type. For example, if all of your entity types are at **second normal form** (2NF) or higher then we say that your data schema is at 2NF.

Table 1: Data Normalisation Rules

Level	Rule
First normal form (1NF)	An entity type is in 1NF when it contains no repeating groups of data.
Second normal form (2NF)	An entity type is in 2NF when it is in 1NF and when all of its non-key attributes are fully dependent on its primary key.
Third normal form (3NF)	An entity type is in 3NF when it is in 2NF and when all of its attributes are directly dependent on the primary key.

The next question is, why data normalisation? The advantages of having a highly normalised data schema are that:

- Information is stored in one place and one place only, reducing the possibility of inconsistent data.
- Furthermore, highly-normalised data schemas in general are closer conceptually to object-oriented schemas because the object-oriented goals of promoting high cohesion and loose coupling between classes results in similar solutions (at least from a data point of view). This generally makes it easier to **map your objects to your data schema**. Unfortunately, normalisation usually comes at a performance cost.

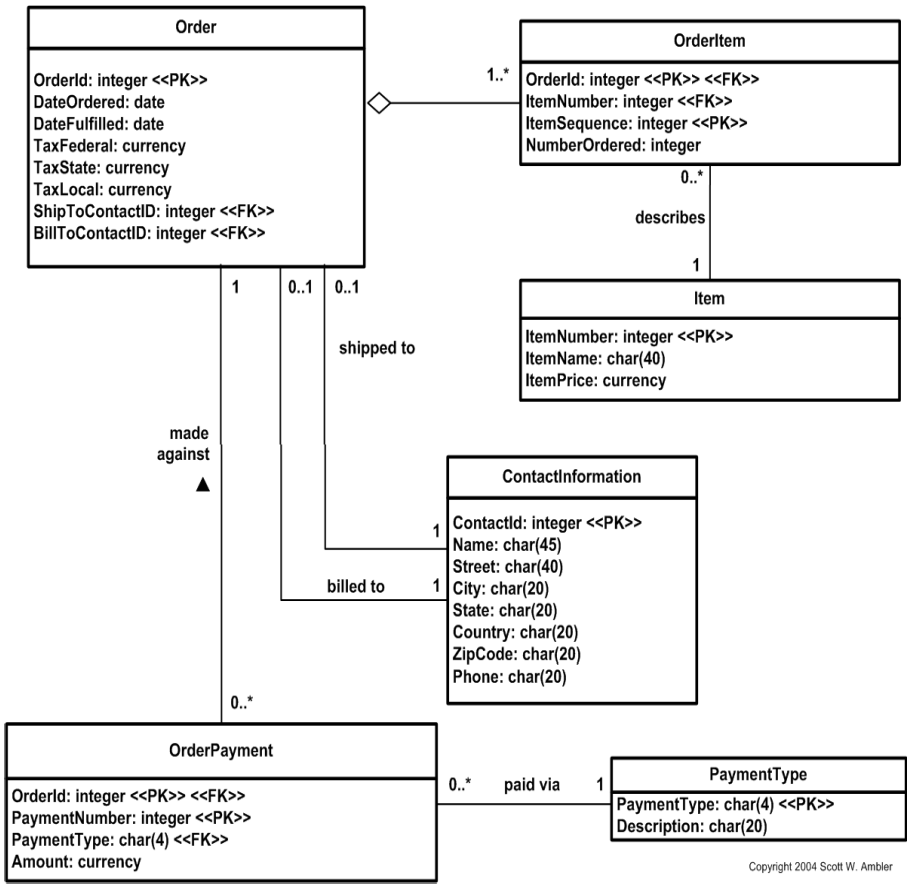


Fig. 7: A Normalised Schema in 3NF (UML Notation)

3.4.8 Denormalise to Improve Performance

Normalised data schemas, when put into production, often suffer from performance problems. This makes sense – the rules of data normalisation focus on reducing data redundancy, not on improving performance of data access. An important part of data modelling is to denormalise portions of your data schema to improve database access times. For example, the data model of Figure 8 looks nothing like the normalised schema of Figure 7.

To understand why the differences between the schemas exist, you must consider the performance needs of the application. The primary goal of this system is to process new orders from online customers as quickly as possible. To do this customers need to be able to search for items and add them to their order quickly, remove items from their order if need be, then have their final order totalled and recorded quickly. The secondary goal of the system is to process, ship, and bills the orders afterwards.

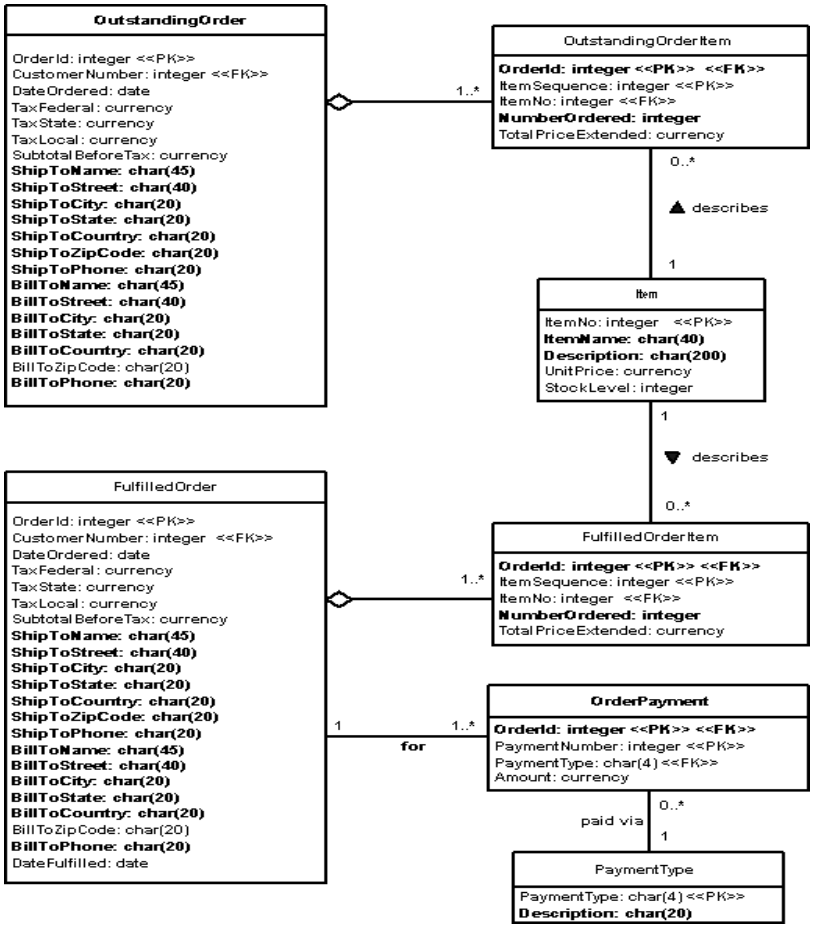


Fig. 8: A Denormalised Order Data Schema (UML notation)

3.5 Advantages and Disadvantages of Data Model:

Advantages of Data model:

- The main goal of a designing data model is to make certain that data objects offered by the functional team are represented accurately.
- The data model should be detailed enough to be used for building the physical database.
- The information in the data model can be used for defining the relationship between tables, primary and foreign keys, and stored procedures.
- Data Model helps business to communicate the within and across organizations.
- Data model helps to documents data mappings in ETL process
- Help to recognize correct sources of data to populate the model

Disadvantages of Data model:

- To develop Data model one should know physical data stored characteristics.
- This is a navigational system produces complex application development, management. Thus, it requires knowledge of the biographical truth.
- Even smaller changes made in structure require modification in the entire application.
- There is no set data manipulation language in DBMS.

3.6 Limitations of Data Models

Data models support data and computer systems by providing the definition and format of data. If this is done consistently across systems, then compatibility of data can be achieved. If the same data structures are used to store and access data, then different applications can share data. However, systems and interfaces often cost more than they should, to build, operate, and maintain.

They may also constrain the business rather than support it. The reason for these problems is a lack of standards that will ensure that, data models will both meet business needs and be consistent. Therefore, the major cause is that, the quality of the data models implemented in systems and interfaces is poor, with the following resultant effects:

- Business rules, specific to how things are done in a particular place, are often fixed in the structure of a data model. This means that small changes in the way business is conducted lead to large changes in computer systems and interfaces.
- Entity types are often not identified, or incorrectly identified. This can lead to replication of data, data structure, and functionality, together with the attendant costs of that duplication in development and maintenance.
- Data models for different systems are arbitrarily different. The result of this is that complex interfaces are required between systems that share data. These interfaces can account for between 25-70% of the cost of current systems.
- Data cannot be shared electronically with customers and suppliers, because the structure and meaning of data has not been standardised. For example, engineering design data and drawings for process plant are still sometimes exchanged on paper.

Case Studies

A school system needed to build a database application (an application where the database is the core component). The problem domain was to track individuals who were being trained to be Health Room Medical Assistants (HMA). The school system has about 140,000 students, exists within an urban setting, and reports to an urban government that exists within a U.S. State.

There are about 210 health rooms across the school system, and whenever the RN (Registered Nurse) is absent from a school, an HMA has to handle the dispensing of medicine, medical record keeping, and first aid. The HMA can only be authorized to perform health-related activities through a “delegating” RN. The HMA “works under” the delegating RN’s license.

School principals are responsible for selecting HMA candidates. They must pass a reading, writing, and arithmetic test. If the test is failed, the HMA candidate cannot come back to the HMA training class for 90 days. The HMA nominee must “sit” for a 40-hour class. There are three tests during the class, and a practical demonstration-based exam at the end. If any test is failed, the HMA candidate is “expelled” and has to retake the entire course starting with the entrance test.

Once an HMA has passed the tests they are deployed. Thereafter, there are follow-up reviews and direct observations every 45 school days. After the first 45-day period, the HMA candidate can apply to the State to become a certified HMA. If after certification the HMA fails a follow-up review and/or any direct observation, the supervising RN must report that failure to the State Board of Nursing. The Board of Nursing may de-certify the HMA. If decertified, the process begins again starting with the entrance test. The rationale for all this is that giving the wrong drugs or dosage to a student, or failing to properly document medical treatments is a serious matter that might cause harm or even death to a student. The State's Board of Nursing takes this seriously.

The task was to build the HMA database application against these requirements. A nurse supervisor of the school system's student health organization chose MS/Access to build the application based on guidance from the school system's IT department. Since the nurse supervisor was told that MS/Access is really simple, and since the nurse supervisor was an HMA subject matter expert, she just fired up Access on her computer and started to type in the database's design: One table, of course. As she discovered something she forgot, she just modified the database's one table design, one column at a time.

After about nine months, the one-table database in MS/Access was complete. She started to type in data for about 300 records of HMAs in various stages of the certification process. Of course "she" herself was required to be present in order to understand and explain each and every row of data and all the implied relationships among the columns. There were multiple status codes, some of which are related to others. There was no history.

During a review of the database's design, and in light of the requirements stated above, questions were asked about the underlying process, test re-takes, test failures, observations, re-certifications, how HMA moves from one school to the next are recorded, how changes in RN delegation are handled, and the automatic refreshing of names, addresses, and schools from the school system's databases. None of these questions had been accommodated within the database's design.

A question was then asked, "Where's your model of the data?" She stated, "Oh, right here." She showed the MS/Access database table. She was not aware that a model of the data is different from a database's design. After this review, the next two hours were then spent figuring out the model of the data. At the end of this first data model design session about 15-20 entities were identified and related. At this point, she knew that these requirements were beyond what she knew how to do with Access. What was starting to emerge was the need for: 1) a data model that would mirror the real requirements, and 2) an application system to manage the user interface, underlying processes, data entry, errors, relationships, editing, workflow, and reporting.

What should she have done? That's the subject of a short paper. The paper was written as if she had all the necessary tools to accomplish all the steps. Did she have these tools? Would the school system expend the resources to make "simple little applications" like these practically possible?

4.0 Self-Assessment Exercise(s)

Access yourself with the following questions:

- What are data model?
- List and explain the types of data models
- State the steps involved in the task of data modeling
- List the advantages and disadvantages of data models
- List the limitations of data model

5.0 Conclusion

The bulk of work in this unit on how to model data and its exploration gives an insight into how structured data are stored in the data management systems such as relational databases. As earlier indicated the main aim of data models is to support the development of information systems by providing the definition and format of data.

6.0 Summary

In this unit we learnt that:

- a data model is simply a diagram that describes the most important “things” in business environment from a data-centric point of view
- the perspective of data models are conceptual, logical and physical schemas
- what makes a good data includes completeness, non-redundancy, enforcement of the business rule, data reusability, stability and flexibility, communication and elegance, integration, etc
- eight basic steps are involved in how to model data.

7.0 Further Readings

American National Standards Institute (1975). ANSI/X3/SPARC Study Group on Data Base Management Systems; Interim Report. FDT (Bulletin of ACM SIGMOD) 7:2.

Andy, G. (2010). The Enterprise Data Model: A Framework for Enterprise Data Architecture.

"Data Modelling for the Business", Steve Hoberman, Donna Burbank, Chris Bradley, Technics Publications, LLC 2009.

David, C. H. (1996). Data Model Patterns: Conventions of Thought. New York: Dorset House Publishers, Inc.

Janis, A. B. jr. (2007). “From Information Algebra to Enterprise Modelling and Ontologies: A Historical Perspective on Modelling for Information Systems”. *In*: Conceptual Modelling in Information Systems Engineering. John Krogstie et al. (Eds). pp. 1-18.

Len, S. & Paul, A. (2008). The Data Model Resource Book: Universal Patterns for Data Modelling. Volume 3. John Wiley & Sons.

Hills, T. (2019). *NoSQL and SQL data modeling: Bringing together data, semantics, and software*. New York: Technics publications.

Hussein, T., Paulheim, H., Lukosch, S., Ziegler, J., & Calvary, G. (2016). *Semantic models for adaptive interactive systems*. London: Springer.

Qin, Z., & Tang, Y. (2014). *Uncertainty modeling for data mining a label semantics approach*. Hangzhou: Zhejiang Univ. Press.

Tutcher, J. (2015). *Development of semantic data models to support data interoperability in the rail industry*. Birmingham: University of Birmingham.

Uzun, A. (2019). *Semantic Modeling and Enrichment of Mobile and WiFi Network Data*. Cham: Springer International Publishing.
Len, S. (2001). The Data Model Resource Book. Volumes 1/2. John Wiley & Sons.

Matthew, W. & Julian, F. (1999). Developing High Quality Data Models. The European Process Industries STEP Technical Liaison Executive (EPISTLE).

Michael, R. M. (1999). “A Conceptual Data Model of Datum Systems”. National Institute of Standards and Technology. August 1999.

RFC 3444 - On the Difference between Information Models and Data Models.

Steve, H.; Donna, B. & Chris, B. (2009). Data Modelling for the Business. Technics Publications, LLC.

Young, J. W. & Kent, H. K. (1958). “Abstract Formulation of Data Processing Problems”. *In*: Journal of Industrial Engineering. Nov-Dec 1958. 9(6), pp. 471-479.

Unit 2: OVERVIEW OF DATA MODELLING

Contents

1.0 Introduction

Data modelling is a critical skill for IT professionals including someone who is familiar with relational databases but who has no experience in data modelling, such people as database administrators (DBAs), data modellers, business analysts and software developers. It is an essential ingredient of nearly all IT projects. Without a data model there is no blueprint for the design of the database.

Then, there are two important things to keep in mind when learning about and doing data modelling:

- Data modelling is first and foremost a tool for communication. There is no single “right” model. Instead, a valuable model highlights tricky issues, allows users, designers, and implementers to discuss the issues using the same vocabulary, and leads to better design decisions.
- The modelling process is inherently iterative: you create a model, check its assumptions with users, make the necessary changes, and repeat the cycle until you are sure you understand the critical issues.

2.0 Intended Learning Outcomes (ILOs)

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

- describe what data modelling is and why it is required
- discuss types and uses of data modeling
- describe the process of data modelling, with the aid of diagram
- describe the outstanding data modelling methodologies
- explain the various data modeling techniques
- explain the properties of data.

3.0 Main Content

3.1 Definition of Data Modelling

Data modelling is the process of creating and extending data models which are visual representations of data and its organisation. The **ERD Diagram** (Entity Relationship Diagram) is the most popular type of data model. In software engineering, it is the process of creating a data model by applying formal data model descriptions using data modelling techniques.

It is a method used to define and analyse the data requirements needed to support the business processes of an organisation. Data modelling defines not just data elements, but their structures and relationships between them. Data modelling is also a technique for detailing business requirements for a database, and it is sometimes called database modelling because, a data model is eventually implemented in a database.

3.2 Types of Data Modelling

Data modelling may be performed during various types of projects and in multiple phases of projects. Data models are progressive; there is no such thing as the final data model for a business or application. Instead a data model should be considered a living document that will change in response to a changing business. The data models should ideally be stored in repository so that they can be retrieved, expanded, and edited over time. Whitten (2004) determined two types of data modelling:

a. **Strategic data modelling**: This is part of the creation of an information systems strategy, which defines an overall vision and architecture for which, information systems is defined. Information engineering is a methodology that embraces this approach.
b. **Data modelling during systems analysis**: In systems analysis logical data models are created as part of the development of new databases.

3.3 Use of Data Modelling

Data modelling techniques and methodologies are used to model data in a standard, consistent, predictable manner in order to manage it as a resource. The use of data modelling standards is

strongly recommended for all projects requiring a standard means of defining and analysing data within an organisation, e.g., using data modelling:
To manage data as a resource
For the integration of information systems
For designing databases/data warehouses (a.k.a data repositories)

3.4 Data Modelling Process

The actual database design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity. The term database design can be used to describe many different parts of the design of an overall database system.
Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and views. In an object database the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the forms and queries used as part of the overall database application within the Database Management System or DBMS.

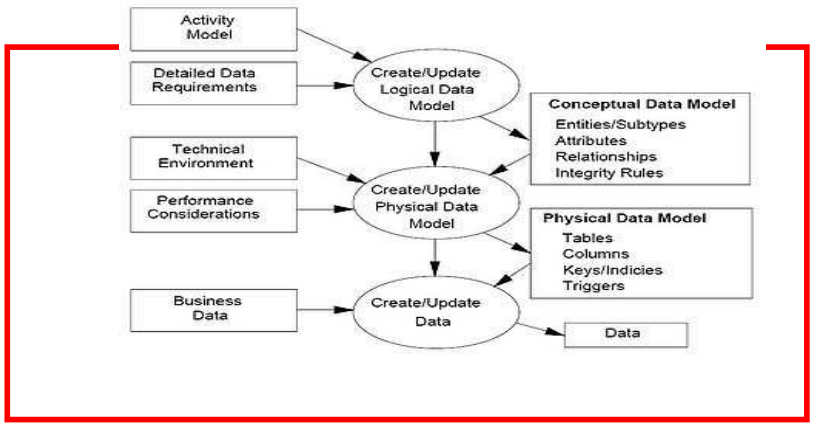


Fig.9: The Data Modelling Process

The figure illustrates the way data models are developed and used today. A conceptual data model is developed based on the data requirements for the application that is being developed, perhaps in the context of an activity model. The data model will normally consist of entity types, attributes, relationships, integrity rules, and the definitions of those objects. This is then used as the start point for interface or database design.

3.5 Modelling Methodologies

Though data models represent information areas of interest, and there are many ways to create data models, according to Len Silverston (1997), only two modelling methodologies stand out:

a. **Bottom-up models:** These are often the result of a reengineering effort. They usually start with existing data structures forms, fields on application screens, or reports. These models are usually physical, application-specific, and incomplete from an enterprise perspective. They may not promote data sharing, especially if they are built without reference to other parts of the organisation.
b. **Top-down logical data models:** These on the other hand, are created in an abstract way by getting information from people who know the subject area. A system may not implement all the entities in a logical model, but the model serves as a reference point or template.

Sometimes models are created in a mixture of the two methods; by considering the data needs and structure of an application and by consistently referencing a subject-area model. Unfortunately, in many environments the distinction between a logical data model and a physical data model is blurred. In addition, some CASE tools don't make a distinction between logical and physical data models.

3.6 Data Modelling Techniques

3.6.1 Flat Data Model

Flat data model is the first and foremost introduced model and in this all the data used is kept in the same plane. Since it was used earlier this model was not so scientific.

Roll No	Name	Course
5482	Mark	Web Designing
5486	Steve	Java
5496	Smith	Oracle

Fig.10: Flat Data Model

Here all members of a column are assumed to be of similar values; all members of a row are assumed to be related to one another.

3.6.2 Hierarchical model

As the name indicates, this data model makes use of hierarchy to structure the data in a tree-like format. Each entity has only one parent but can have several children. At the top of the hierarchy, there is one entity, which is called the root. However, retrieving and accessing data is difficult in a hierarchical database. This is why it is rarely used now.

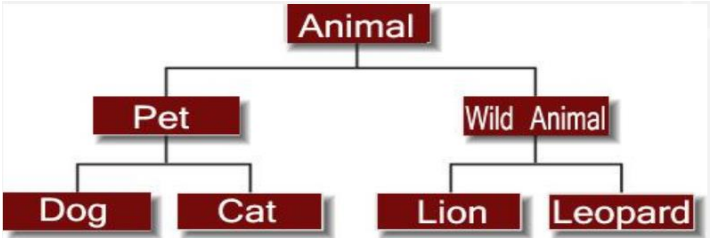


Fig.10: Hierarchical Data Model

3.6.3 Network model

The network model is inspired by the hierarchical model. However, unlike the hierarchical model, this model makes it easier to convey complex relationships as each record can be linked with multiple parent records. Here, entities can be accessed through several path

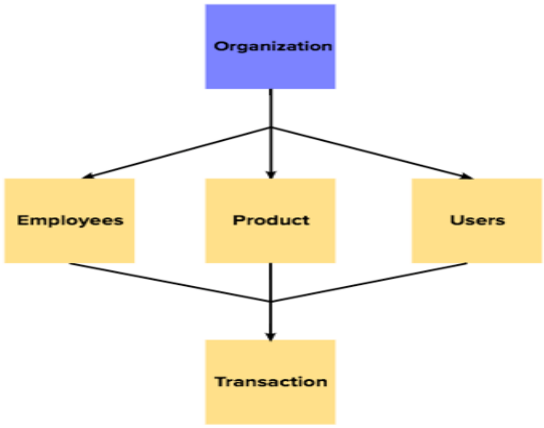
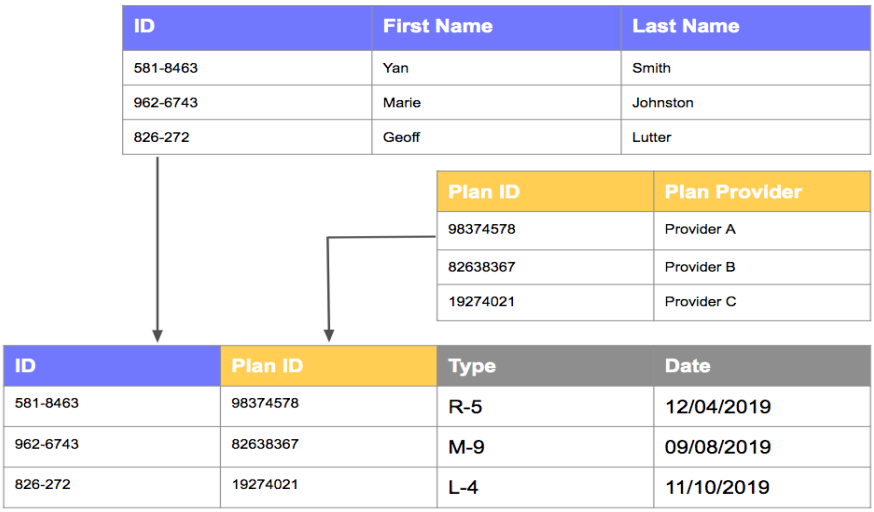


Fig.11: Network Data Model

3.6.4 Relational Model

Proposed as an alternative to hierarchical model by an IBM researcher, here data is represented in the form of tables (two dimensional) called relations. Relational model is the most popular model and the most extensively used model. It reduces the complexity and provides a clear overview of the data.



ID	First Name	Last Name
581-8463	Yan	Smith
962-6743	Marie	Johnston
826-272	Geoff	Lutter

Plan ID	Plan Provider
98374578	Provider A
82638367	Provider B
19274021	Provider C

ID	Plan ID	Type	Date
581-8463	98374578	R-5	12/04/2019
962-6743	82638367	M-9	09/08/2019
826-272	19274021	L-4	11/10/2019

Fig.12: Relational Data Model

3.6.5 Object-oriented model

This database model consists of a collection of objects, each with its own features and methods. This type of database model is also called the post-relational database model. This can hold the audio, video and graphic files.

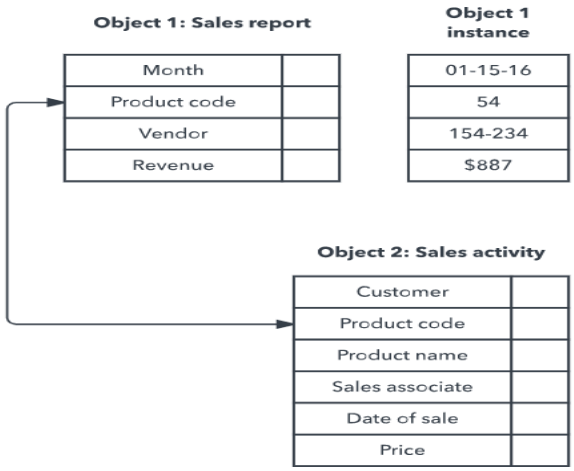


Fig.13: Object-oriented Data Model

3.6.6 Entity-relationship model

Entity-relationship model, also known as ER model, represents entities and their relationships in a graphical format. An entity could be anything – a concept, a piece of data, or an object. This model is dependent on two vital things and they are: (i) Entity and their attributes (ii) Relationships among entities

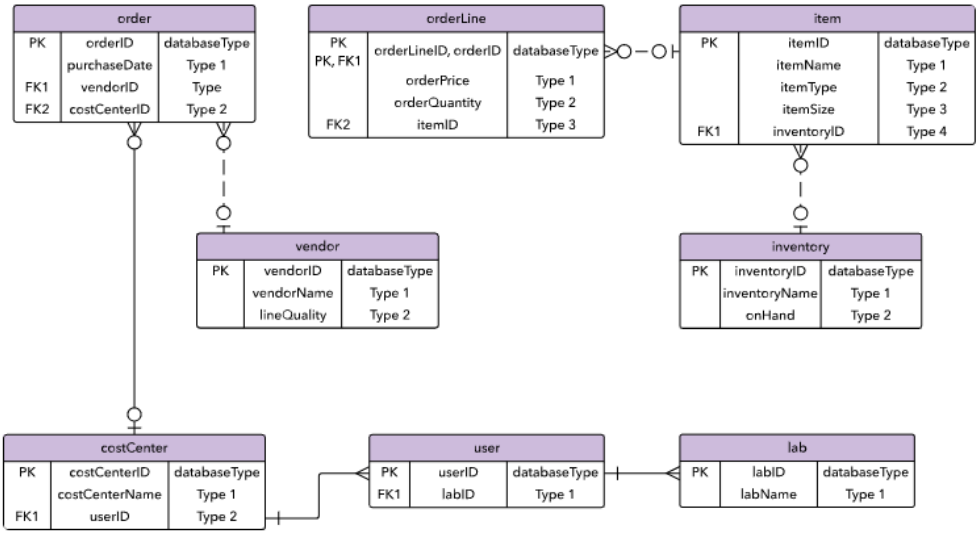


Fig.14: Entity-Relationship Data Model

3.7 Benefits of Data Modelling

Abstraction: The act of abstraction expresses a concept in its minimum, most universal set of properties. A well abstracted data model will be economical and flexible to maintain and enhance accuracy, since it will utilise few symbols to represent a large body of design. If we can make a general design statement which is true for a broad class of situations, then we do not need to recode that point for each instance. We save repetitive labour; minimise multiple opportunities for human error; and enable broad scale, uniform change of behaviour by making central changes to the abstract definition.

In data modelling, strong methodologies and tools provide several powerful techniques which support abstraction. For example, a symbolic relationship between entities need not specify details of foreign keys since they are merely a function of their relationship. Entity sub-types enable the model to reflect real world hierarchies with minimum notation. Automatic resolution of many-to-many relationships into the appropriate tables allows the modeller to focus on business meaning and solutions rather than technical implementation.

Transparency: Transparency is the property of being intuitively clear and understandable from any point of view. A good data model enables its designer to perceive truthfulness of design by presenting an understandable picture of inherently complex ideas. The data model can reveal inaccurate grouping of information (normalisation of data items), incorrect relationships between objects (entities), and contrived attempts to force data into preconceived processing arrangements.

It is not sufficient for a data model to exist merely as a single global diagram with all content smashed into little boxes. To provide transparency a data model needs to enable examination in several dimensions and views: diagrams by functional area and by related data structures; lists of data structures by type and groupings; context-bound explosions of details within abstract

symbols; data based queries into the data describing the model.

Effectiveness: An effective data model does the right job - the one for which it was commissioned - and does the job right - accurately, reliably, and economically. It is tuned to enable acceptable performance at an affordable operating cost.
To generate an effective data model the tools and techniques must not only capture a sound conceptual design but also translate into a workable physical database schema. At that level a number of implementation issues (e.g., reducing insert and update times; minimising joins on retrieval without limiting access; simplifying access with views; enforcing referential integrity) which are implicit or ignored at the conceptual level must be addressed.

An effective data model is durable; that is it ensures that a system built on its foundation will meet unanticipated processing requirements for years to come. A durable data model is sufficiently complete that the system does not need constant reconstruction to accommodate new business requirements and processes.

4.0 Self-Assessment Exercise(s)

Access yourself with the following questions

- What do you understand by the term data modelling?
- Explain the modelling processes and mention its methodologies
- List and explain five (5) data modeling technique

5.0 Conclusion

With the overview of data modelling, individuals and organisations can uncover hidden processes, methodologies, as well as the benefits of modelling their data, which they can use to predict the behaviour of customers, products and processes.

6.0 Summary

In this unit, you have learnt that:

- data modelling is a critical skill for IT professionals, and that, it is first and foremost a tool for communication and inherently iterative
- data modelling is the process of creating and extending data models which are visual representations of data and its organization
- some of the uses of data modelling include data management, integration of information systems, and designing of databases
- benefits of data modelling are abstraction, transparency and effectiveness.

7.0 Further Readings

American National Standards Institute (1975). *ANSI/X3/SPARC Study Group on Data Base Management Systems; Interim Report*. FDT (Bulletin of ACM SIGMOD) 7:2.

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.

Data Integration Glossary, U.S. Department of Transportation, August 20.

FIPS Publication 184 Released of IDEF1X by the Computer Systems Laboratory of the National Institute of Standards and Technology (NIST). 21 December 1993.

Importance-of-Logical-Data-Model. http://www.blueink.biz/RapidApplicationDevelopment.aspx

Len, S.; Inmon, W.H. and Kent, G. (2007). *The Data Model Resource Book*. Wiley, (1997). Reviewed by Van Scott on tdan.com. Accessed 1 Nov 2008.

Paul, R. S. & Richard, S. (1993). Creating a Strategic Plan for Configuration Management Using Computer Aided Software Engineering (CASE) Tools. Paper for 1993 National DOE/Contractors and Facilities CAD/CAE User's Group.

Semantic Data Modeling" In: *Metaclasses and their Application*. Book Series Lecture Notes in Computer Science. Publisher Springer Berlin / Heidelberg. Vol. 943/1995.

Whitten, J. L; Lonnie, D. B. and Kevin, C. D. (2004). *Systems Analysis and Design Methods* (6th ed.).

Unit 3: DATA MODELLING CONCEPTS

Contents

1.0 Introduction

The motive of data modelling concepts is that, all developers should have skills that can be applied on project, with the philosophy that, every IT professional should have a basic understanding of data modelling. This is a brief introduction to these skills. So, it is critical for application developers to understand the concepts and appreciate the fundamentals of data modeling.

2.0 Intended Learning Outcomes (ILOs)

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

- describe generic data model
- differentiate between identifier, modifier and descriptor
- explain different concepts of data modeling
- explain relational and entity-relational models

- explain with the aid of diagram, the syntax of common data modelling notations.

3.0 Main Content

3.1 Generic Data Modelling

Generic data models are generalisations of conventional data models. They define standardised general relation types, together with the kinds of things that may be related by such a relation type. The definition of generic data model is similar to the definition of a natural language. For example, a generic data model may define relation types such as a 'classification relation', being a binary relation between an individual thing and a kind of thing (a class) and a 'part-whole relation', being a binary relation between two things, one with the role of part, the other with the role of whole, regardless the kind of things that are related.

Given an extensible list of classes, this allows the classification of any individual thing and to specify part-whole relations for any individual object. By standardisation of an extensible list of relation types, a generic data model enables the expression of an unlimited number of kinds of facts and will approach the capabilities of natural languages. Conventional data models, on the other hand, have a fixed and limited domain scope, because the instantiation (usage) of such a model only allows expressions of kinds of facts that are predefined in the model.

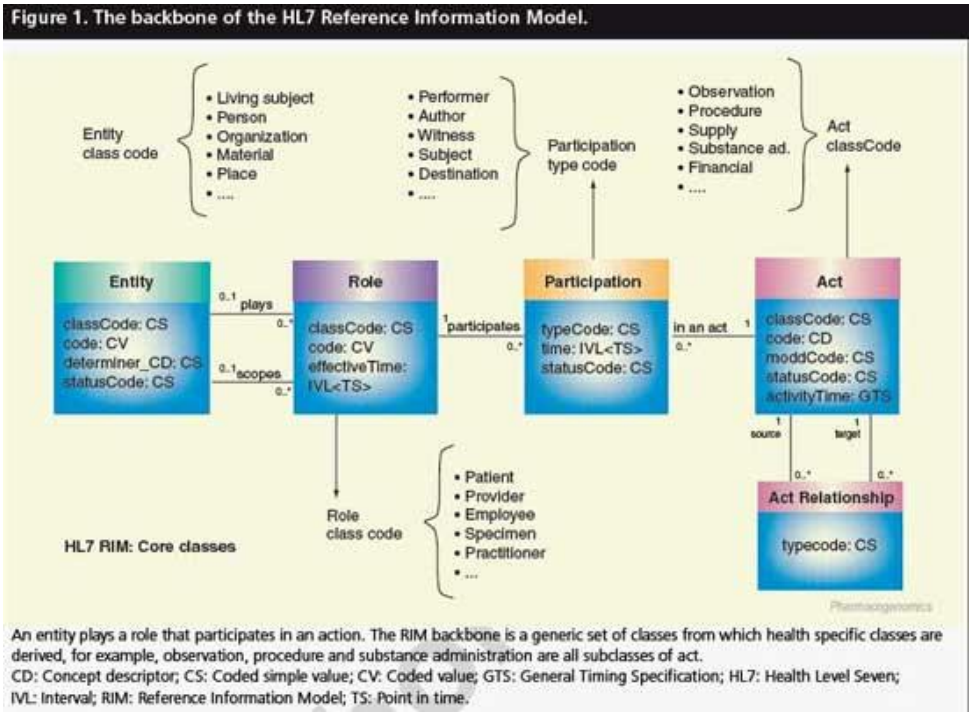


Fig. 14: *Example of a Generic Data Model*

3.2 Concept of Identifier, Modifier, and Descriptor

3.2.1 Identifier (I)

Identifiers serve as the primary identification terms, or keys, necessary to uniquely identify things and events, or classes of them. They symbolically represent things and events (entities) and provide for the necessary identification of conceptual objects. Importantly, identifiers provide the skeletal structure upon which all other types of data depend. They also provide a means for explicitly defining the relationships between things and events (entities), which enables data sharing among users. Typical identifiers include: patient, account, part and purchase order numbers.

3.2.2 Modifier (M)

Modifiers serve as sub-entity identifiers and expand upon or refine primary identification (identifiers). As variant forms of identification, they cannot stand alone. Modifiers must be used in conjunction with identifiers to form fully qualified identification terms. Primarily, they are used to identify such things as: time, occurrence, use, type, sequence, etc. Modifiers have a unique data element value for each variation of the identifier addressed and can exist with one-to-one (1:1) or one-to-many (1:m) cardinality. Typical modifiers include: dates, type codes, serial numbers and revisions.

3.2.3 Descriptor (D)

Descriptors are non-identification data elements used to characterise entities and relationships of them. There are no logical dependencies between descriptors and they can only exist when associated with an identifier or identifier-modifier combination. Descriptors comprise the majority of all data elements and frequently are textual or codified data element values -- data that must be further interpreted by the user to have meaning. Some descriptors are numbers capable of being mathematically manipulated, while others are numerals that do not follow strict mathematical rules. Typical descriptors include: dates, names, descriptions, codes, numeric values *and* images.

3.3 Relational Model and Concept of Relationships

The relational model used the basic concept of a relation or table. The columns or fields in the table identify the attributes such as name, age, and so on. A **tuple** or **row** contains all the data of a single instance of the table such as a person named Doug.

In the relational model, every tuple must have a unique identification or key based on the data. In this figure, a social security account number (SSAN) is the key that uniquely identifies each tuple in the relation. Often, keys are used to join data from two or more relations based on matching identification. The relational model also includes concepts such as foreign keys, which are primary keys in one relation that are kept in another relation to allow for the joining of data.

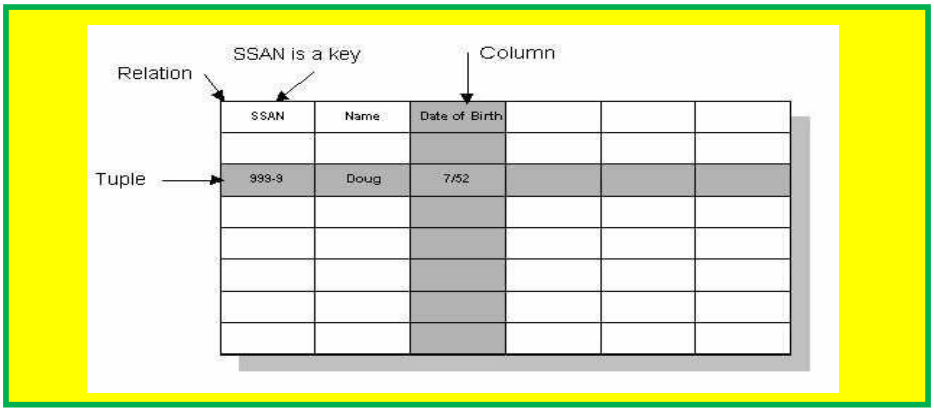


Fig.15: *Relational Model and Relationships*

A **relationship** is a logical connection between two or more entities. Meaningful data in an application includes relationships among its constituent parts. Relationships are essential to data modelling, yet the relational database model does not explicitly support relationships. Instead, primary keys, foreign keys, and referential integrity are used to implement some of the constraints implied by relationships.

In contrast, the Entity Data Model (EDM) provides explicit support for relationships in the data model, which results in flexible modelling capabilities. Relationship support extends to EDM queries, permitting explicit referencing and navigation based on relationships.

3.3.1 Characteristics of Relationships

Relationships are characterised by degree, multiplicity, and direction. In data modelling scenarios, relationships have degree (unary, binary, ternary, or n-ary), and multiplicity (one-to-one, one-to-many, or many-to-many). Direction can be significant in some associations, if, for example, the association is between entities of the same type.

The characteristics of relationships are shown in the following diagrams:

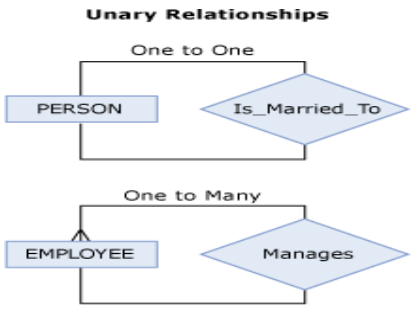


Fig.16a: Unary Relationships

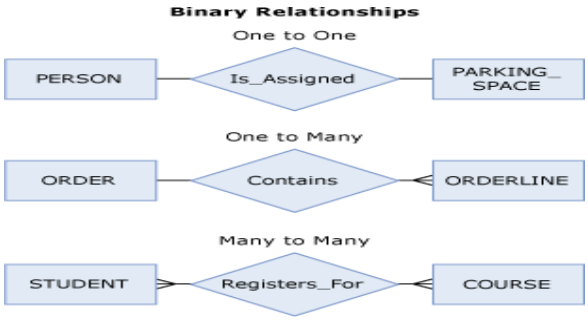


Fig.16b: Binary Relationships

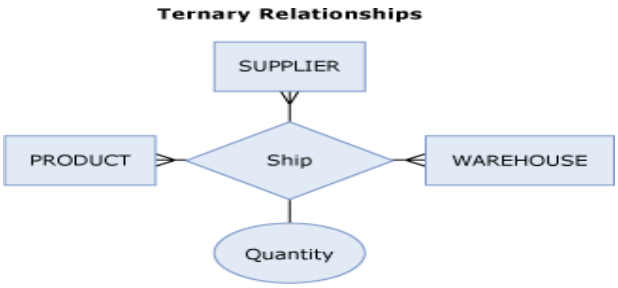


Fig.16c: Ternary Relationships

The degree of the relationship in each diagram is represented by the number of rectangles. Relationships are represented by diamond-shaped figures. The lines between the diamonds and the rectangles represent the multiplicity of the relationships. A single line represents a one-to-one relationship. A line that branches into three segments where it connects to the type represents the many ends of one-to-many or many-to-many relationships.

Degree: The degree of a relationship is the number of types among which the relationship exists. The most common degree of relationship is *binary*, which relates two types. In a unary relationship one instance of a type is related to another instance of the same type, such as the manager relationship between an employee and another employee. A *ternary* relationship relates three types and an *n-ary* relationship relates any number (n) of types. Ternary and n-ary relationships are mainly theoretical. The EDM supports unary and binary relationships.

Multiplicity: Multiplicity is the number of instances of a type that are related. A binary relationship exists between a book and its author, for example, where each book has at least one author. The relationship is specified between the class Book and the class Author, but the multiplicity of this relationship is not necessarily one book to one author. The multiplicity of the relationship indicates the number of authors a book has and the number of books each author has written. The degree of the relationship in this example is binary. The multiplicity of the relationship is many-to- many.

Direction: In the Entity Data Model (EDM), all relationships are inverse relations. An EDM association can be navigated starting from either end. If the entities at the ends of an association

are both of the same type, the role attribute of the EDM association End property can be used to specify directionality. An association between an employee and the employee's manager is semantically different from the two ends of the association. Both ends of the association are employees, but they have different Role attributes.

3.3.2 What is Relational Model?

The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Some popular Relational Database management systems are:

- DB2 and Informix Dynamic Server - IBM
- Oracle and RDB – Oracle
- SQL Server and Access - Microsoft

3.3.3 Relational Model Concepts

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME, etc.
2. **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain

3.3.4 Relational Integrity constraints

Relational Integrity constraints are referred to conditions which must be present for a valid relation. These integrity constraints are derived from the rules in the mini-world that the database represents.

There are many types of integrity constraints. Constraints on the Relational database management system are mostly divided into three main categories are:

1. Domain constraints
2. Key constraints
3. Referential integrity constraints

Domain Constraints: Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Example:

```
Create DOMAIN CustomerName
CHECK (value not NULL)
```

The example shown demonstrates creating a domain constraint such that CustomerName is not NULL

Key constraints: An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

Example:

In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName = "Google".

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Fig.17: Key Constraints

Referential integrity constraints: Referential integrity constraints are based on the concept of Foreign Keys. A foreign key is an important attribute of a relation which should be referred to in other relationships. Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

Example:

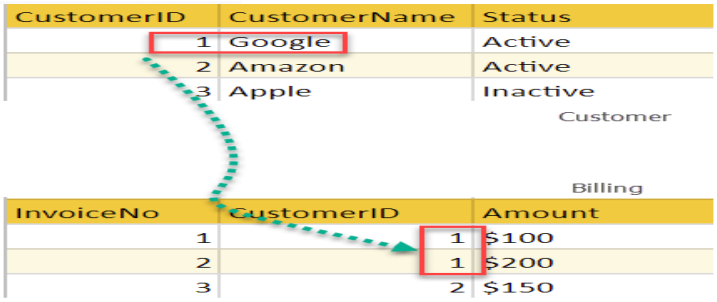


Fig.18: Referential integrity constraints

In the above example, we have 2 relations, Customer and Billing.

Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount $300

3.3.5 Operations in Relational Model

Four basic update operations performed on relational database model are

Insert, update, delete and select.

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

***Insert Operation*:** The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.

Fig.19: Insert operation in relational model

Update Operation: You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.

Fig.20: Update operation in relational model

Delete Operation: To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.

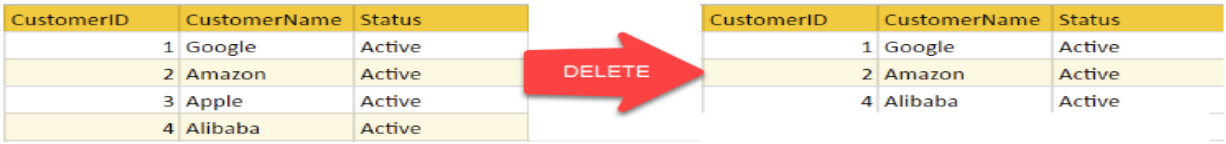


Fig.21: Delete operation in relational model

In the above-given example, CustomerName= "Apple" is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.

Select Operation

Fig.22: Insert operation in relational model

In the above-given example, CustomerName="Amazon" is selected

3.4 Concept of Attributes

The representation of the entity in the data model includes all of the characteristics and attributes of the entity, the actual data elements which must be present to fully describe each characteristic and attribute, and a representation of how that data must be grouped, organised and structured. Although the terms characteristics and attributes can sometimes be used interchangeably, attributes are the more general term, and characteristics are special use attributes.

An Attribute is any aspect, quality, characteristic or descriptor of either an entity or a relationship or may be a very abstract or general category of information, a specific attribute or element, or level of aggregation between these two extremes.

An attribute must also be:

- of interest to the corporation
- capable of being described in real terms, and
- relevant within the context of the specific environment of the firm.

An attribute must be capable of being defined in terms of words or numbers. That is, the attribute must have one or more data elements associated with it. An attribute of an entity might be its name or its relationship to another entity. It may describe what the entity looks like, where it is located, how old it is, how much it weighs, etc. An attribute may describe why a relationship exists, how long it has existed, how long it will exist, or under what conditions it exists.

An attribute is an aspect or quality of an entity which describes it or its actions. An attribute may describe some physical aspect, such as size, weight or colour, or an aspect of the entity's location such as place of residence or place of birth. It may be a quality such as the level of a particular skill, educational degree achieved, or the dollar value of the items represented by an order.

A characteristic is some general grouping of data elements which serve to identify or otherwise

distinguish or set apart one thing or group of things from another. A characteristic is a special form of attribute. It may be a very abstract or general category of information, an element, or level of aggregation between these two extremes. It is also some aspect of the entity that is required to gain a complete understanding of the entity, its general nature, its activities or its usage.

3.5 Entity Concept

An entity type is an abstraction that represents classes of real-world objects. An entity is a Person, Place, Plant, Thing, Event, or Concept of interest to the business or organisation about which data is likely to be kept. For example, in a school environment possible entities might be Student, Instructor, and Class. An entity type refers to a generic class of things such as Company and its property is described by its attribute types and relationship types.

An entity usually has attributes (i.e., data elements) that further describe it. Each attribute is a characteristic of the entity. An entity must possess a set of one or more attributes that uniquely identify it (called a primary key). The entities on an Entity-Relationship Diagram are represented by boxes (i.e., rectangles). The name of the entity is placed inside the box.

Identifying entities is the first step in data modelling. Start by gathering existing information about the organisation. Use documentation that describes the information and functions of the subject area being analysed, and interview subject matter specialists (i.e., end-users).

Derive the preliminary entity-relationship diagram from the information gathered by identifying objects (i.e., entities) for which information is kept. Entities are easy to find. Look for the people, places, things, organisations, concepts, and events that an organisation needs to capture, store, or retrieve.

There are three general categories of entities into one of the following categories

- people, for example, doctor, patient, employee, customer
- fixtures, supplies.
- products, such as goods and services

Conceptual entities are not tangible and are less easily understood. They are often defined in terms of other entity-types. They generally fall into one of the following categories:

- organisations, for example, corporation, church, government,
- agreements, for example, lease, warranty, mortgage,
- abstractions, such as strategy and blueprint.

Event/State entities are typically incidents that happen. They are very abstract and are often modelled in terms of other entity-types as an **Associative entity**. Examples of events are purchase, negotiation, service call, and deposit. Examples of states are ownership, enrolment, and employment

There are also three types of entities:

a. **Fundamental Entities:** These are entities that depict real things (Person, Place, or Concept, Thing etc).
b. **Associative Entities:** These are used for something that is created that joins two entities

(for example, a receipt that exists when a customer and salesperson complete a transaction).

c. **Attributive Entities:** These are used for data that is dependent upon a fundamental entity and are useful for describing attributes (for example, to identify a specific copy of a movie title when a video store has multiple copies of each movie).

3.6 Entity Relationship Model

Structured data is stored in databases. Along with various other constraints, this data's structure can be designed using entity relationship modelling, with the end result being an entity relationship diagram.

Data modelling entails the usage of a notation for the representation of data models. There are several notations for data modelling. The actual model is frequently called "Entity relationship model", because it depicts data in terms of the entities and relationships described in the data.

An entity-relationship model (ERM) is an abstract conceptual representation of structured data. Entity-relationship modelling is a relational schema database modelling method, used in software engineering to produce a type of conceptual data model (or semantic data model) of a system, often a relational database, and its requirements in a top-down fashion.

The elements that make up a system are referred to as **entities**. A **relationship** is the association that describes the interaction between entities.

An **entity-relationship diagram** is a graphical depiction of organisational system elements and the association among the elements. E-R diagrams can help define system boundaries.

An E-R diagram may also indicate the **cardinality** of a relationship. Cardinality is the number of instances of one entity that can, or must, be associated with each instance of another entity. In general we may speak of one-to-one, one-to-many, or many-to-many relationships.

There are several different styles used to draw entity-relationship diagrams. The **Kendall** and **Kendall text** uses the crow's foot notation. Using this notation entities are represented by rectangles and relationships are indicated by lines connecting the rectangles. Cardinality is shown by a series of "tick marks" and "crows feet" superimposed on the relationship lines.

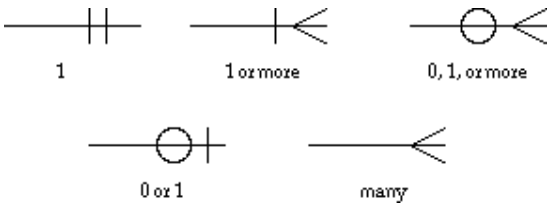


Fig.23:

In the following example each student fills one seat in a class. Each seat is filled by one student. (In this usage a "seat" implies not only a physical place to sit but also a specific day and time.) This is a one-to- one relationship.

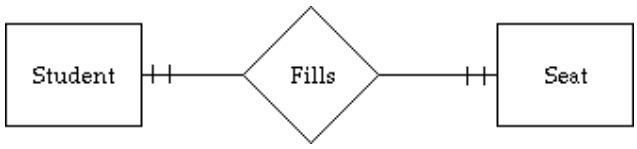


Fig.24

In the next example a single instructor may teach several courses. Each course has only one instructor. This is a one-to-many relationship.

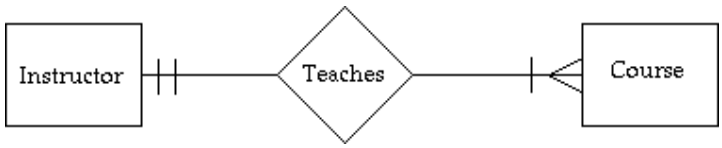


Fig.25

As shown below, a single student may register for several courses. A single course can have many students enrolled in it. This is the many-to- many relationship.

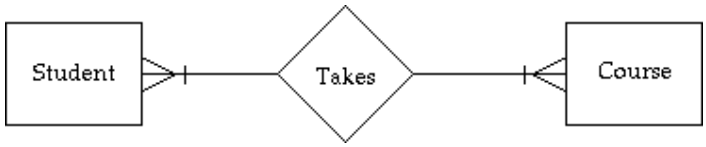


Fig.26

The next example shows a relationship in which it is possible that no instances exist. Each professor may teach several course sections but may not teach at all if on sabbatical. Assume there is no team teaching; therefore each section must have a single professor.

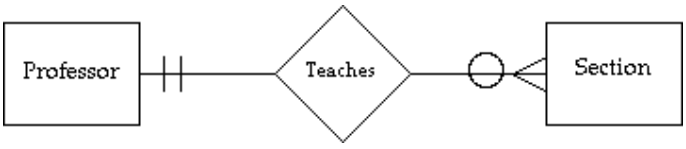


Fig.27

Finally, a more complex example which shows more than one relationship. All of the examples above depict single relationships. An actual E-R diagram would show the many entities and relationships that exist within a system. Here each department offers at least one course; there is no cross-listing of courses with other departments. Each course must have at least one section but often has several sections.

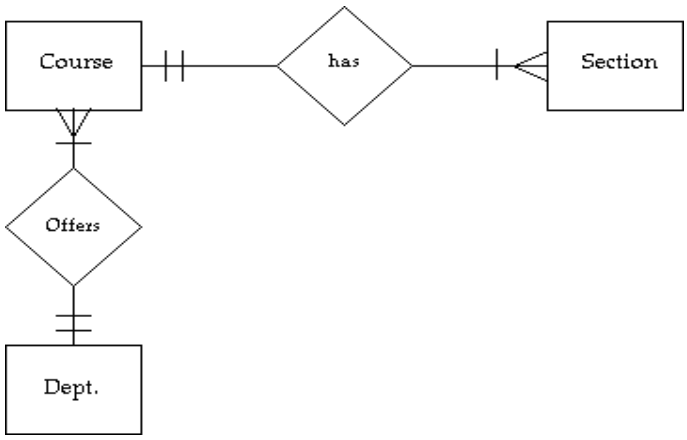


Fig.28

The E-R notation used in the Kendall and Kendall text (4th through 6th editions) also allows for distinguishing different types of entities. A plain rectangle is used for what is termed a **fundamental entity**, that is, an entity that is a real thing (person, place, or thing). The term **associative entity** is used for something that is created that joins two entities (for example, a receipt that exists when a customer and a salesperson complete a transaction). And, the term **attributive entity** is used for data that is dependent upon a fundamental entity and is useful for describing attributes (for example, to identify a specific copy of a movie title when a video store has multiple copies of each movie).

Fig.29

In an entity relationship model, attributes can be composite, derived, or multi-valued.

Multi-valued attributes might have more than one value in one or more instances of its entity. These attributes are denoted with a two line ellipse. So, if a piece of software happens to run on more than one operating system, it could have the attribute "platform"; this is a multi- valued attribute. Composite attributes, on the other hand, are those attributes that might contain two or more attributes.

A composite attribute will have contributing attributes of its own. An example of a composite attribute is an address, as it is composed of attributes that include street address, city, state/region, country, etc

Finally, there are **derived attributes**. The value of these attributes is dependent wholly on another attribute. Derived attributes are denoted with dashed ellipses. So, in a database of employees that has an age attribute, the age attribute would have to be derived from an attribute of birth dates.

These models are being used in the first stage of information system design during the requirements analysis to describe information needs or the type of information that is to be stored in a database. The data modelling technique can be used to describe any ontology (i.e. an overview and classifications of used terms and their relationships) for a certain universe of discourse i.e. area of interest.

3.7 Common Data Modelling Notations

The figure below presents a summary of the syntax of four common data modelling notations: Information Engineering (IE), Barker, IDEF1X, and the Unified Modelling Language (UML). This diagram isn't meant to be comprehensive; instead its goal is to provide a basic overview.

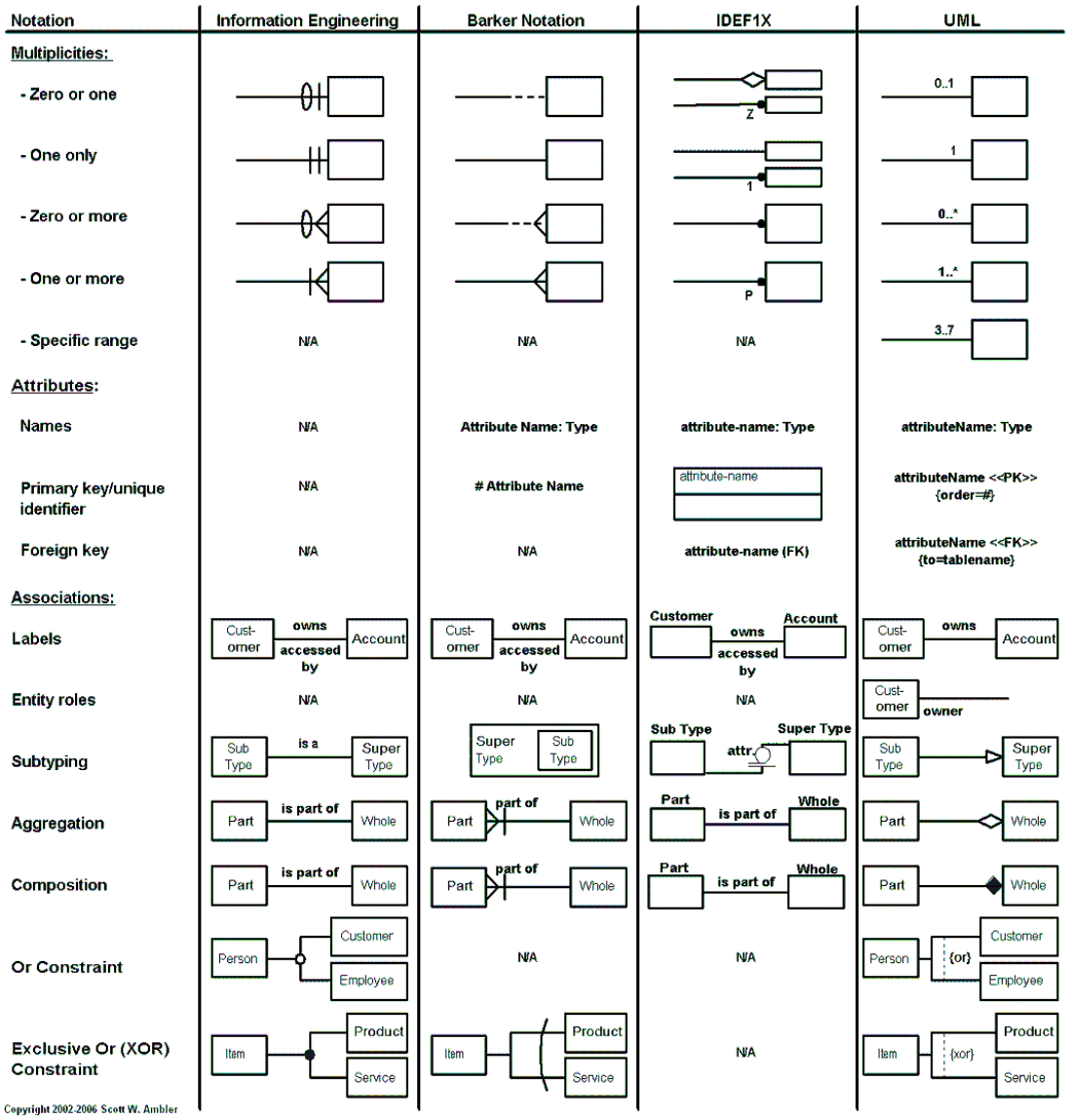


Fig.30 Comparing the Syntax of Common Data Modelling Notations

4.0 Self-Assessment Exercise(s)

Access yourself with the following questions

(1) What is the difference between identifier, modifier and descriptor
(2) Explain different concepts of data modeling
(3) List and explain the operations of a relational model
(4) List and explain the attributes in an entity-relationship model
(5) Identify the syntax of four (4) common data modelling notations

5.0 Conclusion

The brief introduction to the concepts of data modelling helps individuals and organisations acquire knowledge of how physical data are diagrammatically modelled. And for this model to be effective, it must be simple enough to communicate to the end user the data structure required by the database yet detailed enough for the database design to use to create the physical structure. The Entity-Relation Model (ER) is the most common method used to build data models for relational databases.

6.0 Summary

In this unit, you have learnt that:

- generic data models define standardised general relation types, together with the kinds of things that may be related by such a relation type
- the relational model used the basic concept of a relation or table, as a relationship is a logical connection between two or more entities, characterised by degree, multiplicity and direction
- is any aspect, quality, characteristic or descriptor of either an entity or a relationship, while an entity type is an abstraction that represents classes of real-world objects
- an entity-relationship model (ERM) is an abstract conceptual representation of structured data.

7.0 Further Readings

Batini, C.; Ceri, S. Kant & Navathe, B. (1991). Conceptual Database Design: *An Entity Relational Approach*. The Benjamin/Cummings Publishing Company.

Date, C. J. (1990). *An Introduction to Database Systems* (5th ed.). Addison-Wesley.

Fleming, C. C. & Barbara, V.H. (1989). *Handbook of Relational Database Design*. Addison-Wesley.

Hills, T. (2019). *NoSQL and SQL data modeling: Bringing together data, semantics, and software*. New York: Technics publications.

Hussein, T., Paulheim, H., Lukosch, S., Ziegler, J., & Calvary, G. (2016). *Semantic models for adaptive interactive systems*. London: Springer.

Qin, Z., & Tang, Y. (2014). *Uncertainty modeling for data mining a label semantics approach*. Hangzhou: Zhejiang Univ. Press.

Tutcher, J. (2015). *Development of semantic data models to support data interoperability in the rail industry*. Birmingham: University of Birmingham.

Uzun, A. (2019). *Semantic Modeling and Enrichment of Mobile and WiFi Network Data*. Cham: Springer International Publishing.

Kroenke, D. (1983).*Database Processing* (2nd ed.). Science Research Associates.

Martin, E. M. (1992). *Data Analysis, Data Modeling, and Classification*. Martin, J. (1989). *Information Engineering*. Prentice-Hall.

Reingruber, M. C. &William, W. G. (1994).The Data Modelling Handbook: *A Best-Practice Approach to Building Quality Data Models*. John Wiley & Sons, Inc.

Simsion, G. (1994). Data Modelling Essentials: *Analysis, Design, and Innovation*. International Thompson Computer Press.

Teory, T. J. (1994). Database Modelling & Design: *The Basic Principles*. (2nd ed.). Morgan Kaufmann Publishers, Inc.

Ter, Bekke J.H. (1991). *Semantic Data Modelling in Relational Environments.*

Module 2: SEMANTIC DATA MODELLING

Module Introduction

Introduce the module and state the units under the module.

Unit 1: Overview of Semantic Data Modelling
Unit 2: Semantic Data Models
Unit 3: Semantic Data Modelling Concepts

Unit 1: Overview of Semantic Data Modelling

Contents

1.0 Introduction

In the previous units, data modelling and models, as well as their concepts and developments, were explicitly examined. In this unit, Semantic Data Modelling will be dealt with. And for this purpose, we will examine a new data modelling approach based on semantic principles, which results in inherently specified data structures, as a singular word or data item hardly can convey meaning to humans, but in combination with the context, a word gets more meaning.

Also, the logical data structure of a DBMS(Data Base Management System), whether hierarchical, network, or relational, cannot totally satisfy the requirements for a conceptual definition of data because it is limited in scope and biased toward the implementation strategy employed by the DBMS. Therefore, the need to define data from a conceptual view has led to the development of semantic data modelling techniques. That is, techniques to define the meaning of

data within the context of its interrelationships with other data.

Then, in a database environment, the context of data items is mainly defined by structure: a data item or object can have some properties ("horizontal structure"), but can also have relationships ("vertical structure") with other objects. In the relational approach vertical structure is defined by explicit referential constraints, but in the semantic approach structure is defined in an inherent way: a property itself may coincide with a reference to another object. This has important consequences for the semantic data manipulation language.

2.0 Intended Learning Outcomes (ILOs)

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

- define semantic data modeling
- state and explain the principles of semantic data modeling
- explain data integrity rule
- explain additional data restrictions and declarative data derivations.

3.0 Main Content

3.1 Definition of Semantic Data Modelling

Semantic Data Modelling is a technique used to define the meaning of data within the context of its interrelationships with other data. In terms of resources, ideas, events, etc., the real world is symbolically defined within physical data stores. A semantic data model is an abstraction which defines how the stored symbols relate to the real world. Thus, the model must be a true representation of the real world.

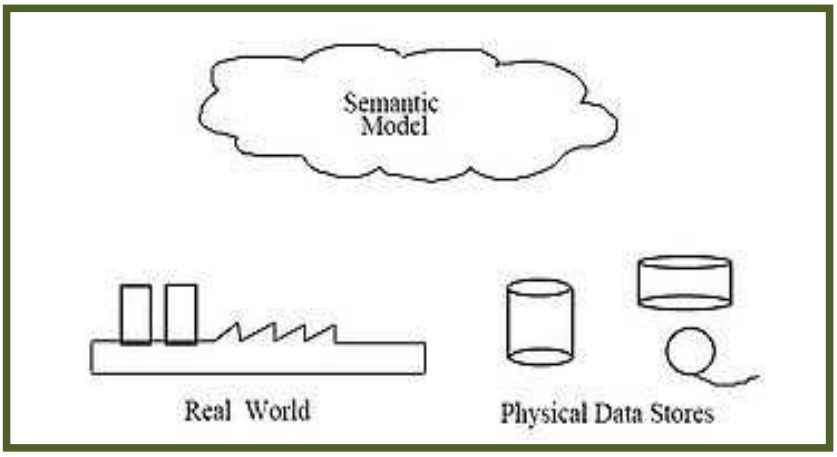


Fig. 31: Semantic Data Models

3.2 Principles of Semantic Data Modelling

The objective of data modelling is to design a data structure for a database fitting as good as possible with some relevant world, often related to an organisation with some information need. In general there

is some relationship between a data model and a part of the existing world, but it is also possible that a data model has a relationship with some imaginary and abstract world. According to Smith and Smith (1977), three abstractions are very important for data modelling:

- Classification
- Aggregation
- Generalisation

a. **Classification** is used to model instance of relations, aggregation to model has a relations and generalisation to model is a relations. In semantic data modelling all three abstractions lead to a type definition (which can be base or composite). The semantic data model is, as contrasted with many other data models, based on only one fundamental notion: the type concept.
It is interesting to know that also the Xplain meta data model requires all three abstractions. Many other data modelling techniques (such as relational model) do not support these three abstractions and therefore are limited in modelling capabilities.
A semantic data model can be represented graphically in an Abstraction Hierarchy diagram showing the types (as boxes) and their inter-relations (as lines). It is hierarchical in the sense that the types which reference other types are always listed above the referenced type. This simple notation principle makes the diagrams very easy to read and understand, even for non-data modellers.

b. **Aggregation**: A type can have attributes (properties), but attributes have a type of their own (for example the type "name") and must be part of a composite type: thus an attribute defines a connection or aggregation of two types. For example, "name" can be an attribute of the type "employee". Using the semantic language, this attribute can be specified as "employee its name". Each composite type has some positive number of attributes, but a **base type** such as "name" does not have any attribute. Within a data model, a type can be identified by a name. At the same time a collection of properties (attributes) identifies a composite type: there may not be two types with the same collection of attributes. Moreover, within a data model a composite type has only one definition: the principle of **convertibility**. Convertibility not only applies to the conceptual level (the data model), but also applies to instances of composite types. Using the following simplified definition of the type "employee", the two mentioned instances of "employee" may not be registered in a same database:
type employee = name, address, town, department.

employee	**name**	**address**	**town**	**Department**
123	John	43 Q Street	Greenfield	12

432	John	43 Q Street	Greenfield	12

Semantic Data Modelling is a technique used to define the meaning of data within the context of its interrelationships with other data. In terms of resources, ideas, events, etc., the real world is symbolically
These two instances are conflicting with the principle of convertibility, but are allowed by an equivalent relational model because relational modelling only requires uniqueness of a primary key value. However, readers might not agree, because a father and his son can have the same shown properties. If we want to deal with such possible situations then it is necessary to extend the definition "employee" with properties such as "birth_date" and "function" or "salary" enabling us to make a distinction between the two instances.
Types are described by their **permanent** properties, in particular the properties relevant for the information needs at hand. If a property or attribute is not relevant to all instances of a certain composite type then the data definition must be improved. The definition of object types related to a same real life object as "person" needs not be the same for all organisations; the FBI and CIA have more information needs than a local administration department.

c. **Generalisation:** In addition to aggregation as a building principle for data modelling, generalisation is another building stone. An example is that a student administration can be interested in different kinds of students: students with or without a job. We could model this as follows:

    type student = name, address, town, birth_date, faculty, employer.

    The problem with this solution is that in the case of nil employer the attribute "student its employer" is not relevant at all. If this model is implemented the people of the student administration department cannot be sure about the relevance of the attribute "student its employer". If they have to insert new instances of "student" the data model as such does not tell them what to do. They can make two kinds of errors: they can ignore the field for employer even if it still is relevant, or they can refer to some employer if it is not relevant. Therefore it is better to design a data model where the type "student" has all the properties that are relevant to all students, which implies that there is not an attribute "student its employer". We call this **generalisation**: the cross section of properties of "student" and "working student" defines the type "student" and the type "working student" is a **specialisation** of "student": it has the additional property "working student its employer":

    type student = name, address, town, birthdates, faculty.

    type working student = [student], employer.

    The square brackets specify that for each instance of "student" there is at most one instance of "working student" and there is one instance of "student" referred to by an instance of "working student". Using generalisation/specialisation it is clear to users which attribute fields are mandatory for registration. Now there is no doubt about the relevance of attribute fields in an application: each attribute field must be filled correctly

and NULL-values are not allowed! The absence of NULL- values is one of the requirements for applying a semantic data manipulation language.

3.3 Data Integrity Rules

Data model specifications imply validity of certain integrity rules. Two inherent integrity rules are recognised for type definitions in a semantic data model:

- **Relatability:** Each attribute in a type definition is related to one and only one equally named type, while each type may correspond with various attributes in other types.
- **Convertibility:** Each type definition is unique: there are no type definitions carrying the same name or the same collection of attributes.

It is important to realise that these two integrity rules require neither separate specification nor declaration by procedures; they are inherent to the type definitions in a semantic data model.

3.4 Additional Data Restrictions

In addition to restrictions inherent from the data model there is often need for additional more complex restrictions on data states that cannot be specified in a data model itself. These additional restrictions can be specified as:

- **Static restrictions** (applicable in all states) are restrictions that apply always. These are defined as assertions. An **assertion** specifies a derivable attribute or a derivable single variable, possibly some value restriction, and the calculation of the derivable item(s). An example is that in many organisations there is only one manager per department:

  assert department its managercount (1..1) = count employee
  where function = manager per department.

  If an organisation wants to guard that the number of employees is not more than 100, the following assertion using a single variable can be specified:

  assert maxemployees (0..100) = count employee.

- **Dynamic restriction** (applicable only in certain states, such as during creation or after update) has to deal with state transitions of data.

3.5 Declarative Data Derivations

The assert command - as explained in static restriction - can be used to specify derivable attributes. This is extremely useful for modelling complex data derivations, such as needed for user applications (e.g. total order amount), reports (e.g. grouping per x, Year-To-Date and totals) and data analysis (top 10 products).An assert command derives only one attribute at a time. For complex derivations you need to define multiple assertions building on each other. This principle ensures modularity (thus easy to understand, test and maintain) and reusability (of intermediate derived data) for other queries.

4.0 Self-Assessment Exercise(s)

1. Define semantic data modeling.
2. Explain the principles of semantic data modelling.
3. Explain static and dynamic data restrictions.

5.0 Conclusion

With semantic data modelling has the meaning of data within the context of its interrelationships with other data. So, its knowledge will help individuals in having understanding of how data are ethically modelled, for the benefit of an enterprise.

6.0 Summary

In this unit, you have learnt that:

- Semantic Data Modelling is a technique used to define the meaning of data within the context of its interrelationships with other data
- the principles of semantic data modelling include classification, aggregation, and generalization
- the data model specifications imply validity of certain integrity rules such as relatability and convertibility
- the additional restrictions can be specified as; static and dynamic restrictions.

7.0 Further Readings

Jack, C. & Colin, K. "Semantic Data Models." Unpublished essay. School of Computing, Napier University. 15 Oct. 2006. <http://www.soc.napier.ac.uk/module.php3

Object Role Modelling: An Overview (msdn.microsoft.com). Retrieved 19 September 2008.

Grady, B.; Ivar, J. & Jim, R. (2000). OMG Unified Modelling Language Specification, Version 1.3 First Edition: March 2000. Retrieved 12 August 2008.

Halpin, T.A. (2001a). *Information Modeling and Relational Databases*. San Francisco: Morgan Kaufmann Publishers. (www.mkp.com/books_catalog/catalog.asp?ISBN=1-55860-672- 6).

Len, S. (2001). *The Data Model Resource Book.* Volumes 1/2. John Wiley & Sons.

Semantic Data Model. 13 Aug. 2001. COMTECO Ltd. 15 Oct. 2006

<http://www.comteco.ru/EN/DATAMODE/datamode.htm

Hills, T. (2019). *NoSQL and SQL data modeling: Bringing together data, semantics, and software*. New York: Technics publications.

Hussein, T., Paulheim, H., Lukosch, S., Ziegler, J., & Calvary, G. (2016). *Semantic models for adaptive interactive systems*. London: Springer.

Qin, Z., & Tang, Y. (2014). *Uncertainty modeling for data mining a label semantics approach*. Hangzhou: Zhejiang Univ. Press.

Tutcher, J. (2015). *Development of semantic data models to support data interoperability in the rail industry*. Birmingham: University of Birmingham.

Uzun, A. (2019). *Semantic Modeling and Enrichment of Mobile and WiFi Network Data*. Cham: Springer International Publishing.

Unit 2: SEMANTIC DATA MODELS

Contents

1.0 Introduction

According to Klas and Schrefl (1995), the "overall goal of semantic data models is to capture more meaning of data, by integrating relational concepts with more powerful abstraction concepts known from

the Artificial Intelligence field. The idea is to provide high level modelling primitives as integral part of a data model, in order to facilitate the representation of real world situations.

2.0 Intended Learning Outcomes (ILOs)

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

- explain what semantic data model is all about
- state the basic requirement of semantic data model
- mention the areas of application of semantic data model
- explain the advantages of semantic data mode over conventional data model

3.0 Main Content

3.1 Overview of Semantic Data Models

The logical data structure of a database management system (DBMS), whether hierarchical, network, or relational, cannot totally satisfy the requirements for a conceptual definition of data, because it is limited in scope and biased toward the implementation strategy employed by the DBMS. Therefore, the need to define data from a conceptual view has led to the development of semantic data modelling techniques. That is, techniques to define the meaning of data within the context of its interrelationships with other data.

The real world, in terms of resources, ideas, events, etc., is symbolically defined within physical data stores. A semantic data model is an abstraction which defines how the stored symbols relate to the real world. Thus, the model must be a true representation of the real world.

3.2 Definition of Semantic Data Models

A semantic data model in software engineering has various meanings:

- It is a conceptual data model in which semantic information is included. This means that the model describes the meaning of its instances. Such a semantic data model is an abstraction that defines how the stored symbols (the instance data) relate to the real world.
- It is a conceptual data model that includes the capability to express information that enables parties to the information exchange, to interpret meaning (semantics) from the instances, without the need to know the meta-model. Such semantic models are fact oriented (as opposed to object oriented). Facts are typically expressed by binary relations between data elements, whereas higher order relations are expressed as collections of binary relations.
- Typically binary relations have the form of triples: Object- Relation Type-Object. For example: the Eiffel Tower <is located in> Paris. Typically the instance data of semantic data models explicitly include the kinds of relationships between the various data elements, such as <is located in>. To interpret the meaning of the facts from the

instances, it is required that the meaning of the kinds of relations (relation types) is known. Therefore, semantic data models typically standardise such relation types. This means that the second kind of semantic data models enable that, the instances express facts that include their own meaning. The second kind of semantic data models are usually meant to create semantic databases. The ability to include meaning in semantic databases facilitates building distributed databases that enable applications to interpret the meaning from the content. This implies that semantic databases can be integrated when they use the same (standard) relation types. This also implies that in general they have a wider applicability than relational or object oriented databases.

3.3 History of Semantic Data Models

The need for semantic data models was first recognised by the U.S. Air Force in the mid-1970s as a result of the Integrated Computer-Aided Manufacturing (ICAM) Program. The objective of this program was to increase manufacturing productivity through the systematic application of computer technology. The ICAM Program identified a need for better analysis and communication techniques for people involved in improving manufacturing productivity. As a result, the ICAM Program developed a series of techniques known as the IDEF (ICAM DEFinition) Methods which included the following:

- **IDEF0** used to produce a "function model" which is a structured representation of the activities or processes within the environment or system.
- **IDEF1** used to produce an "information model" which represents the structure and semantics of information within the environment or system.
- **IDEF1X** is a semantic data modelling technique. It is used to produce a graphical information model which represents the structure and semantics of information within an environment or system. Use of this standard permits the construction of semantic data models which may serve to support the management of data as a resource, the integration of information systems, and the building of computer databases
- **IDEF2** used to produce a "dynamics model" which represents the time varying behavioural characteristics of the environment or system.

3.4 Semantic Data Model Requirements

The data model should meet the following requirements**:**

- allow to specify well-defined schemata (schema definition language)
- support dynamic schema evolution to capture new or evolving types of semantic information
- be simple to use, light-weight, make no assumptions about the semantics of the metadata
- be platform independent and provide interoperability between applications that manage and exchange metadata
- facilitate integration with resources outside the file store and support exporting metadata to the web
- leverage existing standards and corresponding tools, such as query languages.

3.5 Advantages of Semantic Data Model over Conventional Data Model

The SDM addresses many, of the problems that are encountered in attempting to construct a meaningful application model with conventional data models. An analysis of the semantic inadequacies of these models highlights the advantages of the SDM. Some of the principal problems of semantic modelling in the relational data model can be summarized as follows:

1. The relational data model is based on syntactic representational data structures. The data base designer or user must effect a conscious transformation from his view of application oriented entities to their representations. In other words, the user is forced to think in terms of data rather than reality; he must deal with the representations of things (or with information about them), rather than with the things themselves. The SDM attacks this problem by directly modelling higher level structures such as concrete objects, point events, abstractions, and so forth. The SDM makes it possible to capture the statics as well as the dynamics of an application environment in a straightforward manner.
2. The relational data model suffers from semantic overloading. The same structure, the relation, is used to express many different kinds of information. The relation models the existence of entities, their properties, their structure, and associations among them. To address this problem, the SDM provides a spectrum of features, each of which is used for a single semantic purpose.
3. In reality, the values of all entries in a relational data base are names, not entitles in the application environment. This name orientation has several implications:
    a. To find a relevant entity given another, it is necessary to obtain the name of the second entity and use it to locate the entity. This unnatural extra step of indirection is avoided in the SDM by allowing entities to stand for themselves, e.g., as values of attributes.
    b. It is difficult to naturally integrate semantic integrity constraints into a relational data base, since they must be phrased in terms of names. For example, it is difficult to restrict the values that some column of ‘a relation can assume to just those representing a particular kind of entity.
    c. For reasons of economy, it becomes necessary to organize the data base so that all references to an entity are made through a standard attribute (often called a “key”). This makes it difficult to use other identifiers in establishing inter-entity connections. Even worse, it is sometimes necessary to introduce a ‘fictional’ property solely for the purposes of Identification. In addition, changing the value of the key of an entity becomes an operation with profound implications. These difficulties stem from a failure to recognize the critical importance of the distinction between an entity and its name; it is well known to programming language designers that failing to adequately accommodate this distinction can be a source of much confusion.
4. The relational model gives rise to data bases with disjoint structures. Relations and the

information in them are Isolated, and are not well interconnected. Tuples of relations can be linked only by join connections, severely limiting the possible ways of expressing interrelationships among entities. An important consequence of this semantic inflexibility is that tuples are members of only one relation. This means, for example, that it is difficult to capture the fact that a tuple in the relation, for all ships and a tuple in the relation for all oil tankers represent the same entity (ship). Furthermore, second order entities (such as abstractions and aggregates) are not directly related to their underlying instances and constituents.

5. Much information about the structure of an application system simply cannot be expressed in a relational schema. The relational data model provides only a limited set of mechanisms with which to express information about the organization of the problem domain, and some of these are cumbersome to use. Knowledge that cannot be naturally represented in these terms must either be carried explicitly as data in the data base, or be captured by means of an external mechanism such as semantic integrity constraints, or it will simply remain unexpressed. The rich set of features provided by the SDM enables much of this knowledge to be directly stated in the schema. We argue that the more information about the meaning and structure of a data base that is explicitly expressed in its schema, the less the user needs to remember and keep in his mind; user interaction with the data base then becomes simmer and more reliable. The situation is analogous to the move from representation to abstraction in data types as n means of achieving software reliability.

3.6 Applications of Semantic Data Models

A semantic data model can be used to serve many purposes, including:

- **Planning of Data Resources:** A preliminary data model can be used to provide an overall view of the data required to run an enterprise. The model can then be analysed to identify and scope projects to build shared data resources.
- **Building of Shareable Databases:** A fully developed model can be used to define an application independent view of data which can be validated by users and then transformed into a physical database design for any of the various DBMS technologies. In addition to generating databases which are consistent and shareable, development costs can be drastically reduced through data modelling.
- **Evaluation of Vendor Software:** Since a data model actually represents the infrastructure of an organisation, vendor software can be evaluated against a company's data model in order to identify possible inconsistencies between the infrastructure implied by the software and the way the company actually does business.
- **Integration of Existing Databases:** By defining the contents of existing databases with semantic data models, an integrated data definition can be derived. With the proper technology, the resulting conceptual schema can be used to control transaction processing in a distributed database environment. The U.S. Air Force Integrated Information Support

System (I2S2) is an experimental development and demonstration of this type of technology applied to a heterogeneous DBMS environment.

- **Formal Specification Language:** The SDM can be used as a formal specification language, for precisely describing the structure of an application system. The specification process is widely recognized as a major component of the total software system design process, and it is one that is often performed poorly. Natural language is notoriously ambiguous for use as a formal specification tool, while lower level representations (such as conventional data models) do not provide the level of abstraction needed to support effective specifications. A formal description expressed in terms of the SDM can serve as a communications medium between an application specialist and a data base designer, precisely describing the system for which a data base is to be constructed. It can also be included in- the permanent documentation of the data base, to provide a meaningful and unambiguous guide to it.

4.0 Self-Assessment Exercise(s)

Access yourself with the following questions:

1. What do you understand by the term semantic data models?
2. Mention and explain the applications of semantic data models.
3. State the advantages of Semantic data models over conventional data model

5.0 Conclusion

The exploration of this unit has clearly revealed to individuals and organisations, how stored symbols relate to the real world, that is, how models depict the true representation of the real world.

6.0 Summary

In this unit, you learnt that:

- a semantic data model is an abstraction which defines how the stored symbols relate to the real world. Thus, the model must be a true representation of the real world
- the applications of semantic data models include planning of data resources, building of shareable databases, evaluation of vendor software, and integration of existing databases
- semantic introduction to databases defines fundamental concepts of databases, which are described in terms of semantic binary model.

7.0 Further Readings

Jack, C. & Colin, K. “Semantic Data Models.” Unpublished Essay. School of Computing, Napier University. 15 Oct. 2006. <http://www.soc.napier.ac.uk/module.php3

Semantic Data Model. 13 Aug. 2001. COMTECO Ltd. 15 Oct. 2006 <http://www.comteco.ru/EN/DATAMODE/datamode.htm
http://en.wikipedia.org/wiki/Category:Wikipedia_articles_incorp
orating_text_from_the_National_Institute_of_Standards_and_Technology

Tina, Y. L. (1999). “Information modeling from design to implementation” National Institute of Standards and Technology.

Hills, T. (2019). *NoSQL and SQL data modeling: Bringing together data, semantics, and software*. New York: Technics publications.

Hussein, T., Paulheim, H., Lukosch, S., Ziegler, J., & Calvary, G. (2016). *Semantic models for adaptive interactive systems*. London: Springer.

Qin, Z., & Tang, Y. (2014). *Uncertainty modeling for data mining a label semantics approach*. Hangzhou: Zhejiang Univ. Press.

Tutcher, J. (2015). *Development of semantic data models to support data interoperability in the rail industry*. Birmingham: University of Birmingham.

Uzun, A. (2019). *Semantic Modeling and Enrichment of Mobile and WiFi Network Data*. Cham: Springer International Publishing.

Unit 3: SEMANTIC DATA MODELLING CONCEPTS

Contents

1.0 Introduction

Typically the instance data of semantic data models explicitly include the kinds of relationships between the various data elements, such as <is located in>. To interpret the meaning of the facts from the instances, it is required that the meaning of the kinds of relations (relation types) is known. Therefore, semantic data models typically standardise such relation types. This means that the second kind of semantic data models enable that the instances express facts that include their own meaning.

The second kind of semantic data models are usually meant to create semantic databases. The ability to include meaning in semantic databases facilitates building distributed databases that enable applications to interpret the meaning from the content. This implies that semantic databases can be integrated when they use the same (standard) relation types. This also implies

that in general they have a wider applicability than relational or object oriented databases.

2.0 Intended Learning Outcomes (ILOs)

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

- explain the essential element of semantic data language
- mention different organisations of semantic models
- describe Semantic Data in the Database
- explain Semantic Annotation, Indexing, and Retrieval
- explain the concepts of entities, design and indexing in semantic modelling.

3.0 Main Content

3.1 Overview of Semantic Data Language

A new and efficient data language (more suitable for end users than SQL) respecting data model structure and avoiding the pitfalls (joins and NULLs) of SQL, was developed by **Johan ter Bekke**. Here is an overview of the main data query and manipulation commands:

Command	Description
Get	to retrieve data
Extend	to derive/compute intermediate data
Value	to store one (possibly derived) value
Insert	to add a record
Update	to change data
Cascade	to process data in a recursive way
Delete	to remove data

Relational join pitfall

Before discussing semantic data language principles it is essential to discuss a main weakness of the relational language SQL leading to a pitfall, especially for non-expert users. In particular we discuss the problems generated by the relational join operation when combined with GROUP BY. In order to be able to discuss that pitfall we use the following simple data model:

- employee (emp#, name, address, town, birth_date, function, dept#);
- department (dept#, address, town, main_task);

The semantic equivalent of this model:

- *type employee = name, address, town, birth_date, function, department.*
- *type department = address, town, main_task.*

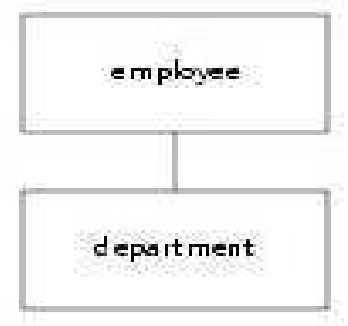


Fig. 33

We suppose that this data model is used by an organisation located in Abuja wanting to determine which departments (dept#) have the smallest number of employees living in Abuja. This information can be relevant when departments offer a compensation for the travelling costs of their employees. A user could think that this problem is solved by the following SQL-query:

```
CREATE VIEW temporal (dept#, number)
AS SELECT department.dept#, COUNT
(emp#) FROM employee, department
WHERE employee.dept# = department.dept#
AND employee.town = Abuja
GROUPBY department.dept#;
SELECT dept#
FROM temporal
WHERE number IN (SELECT MIN (number) FROM temporal);
```

If a department does not have any employee living in Abuja, its dept# will not be present in the temporal table created by the view temporal. A more extended query solves this problem, using a union of two sub sets:

```
CREATE VIEW temporal (dept#, number) AS
(SELECT department.dept#, COUNT (emp#)
FROM employee, department
WHERE employee.dept# = department.dept#
AND employee.town = Abuja
GROUP BY department.dept#)
UNION(SELECT dept#, 0 FROM department WHERE dept#
NOT IN (SELECT dept# FROM employee WHERE town = Abuja));
SELECT dept# FROM temporal WHERE number IN (SELECT
MIN (number) FROM temporal);
```

The last solution guarantees that, if a department does not have any employee living in Abuja the value of the derived attribute temporal number is 0 for such a department. Now the derived information will be correct (and complete) irrespective the contents of the database.

The semantic language does not allow for join-operations; it requires applying paths really existing in the underlying data model. The first step is to derive the required number of employees per department, using the temporal attribute department its number:
extend department *with* number = *count* employee
where town = Abuja *per* department.
value minimum = *min* department *its* number.
get department *where* number = minimum.

The term per means the same as for each, which most probably is easier to understand then GROUP BY: if an empty sub set of data is involved in a join combined with the GROUP BY construct then an incomplete, thus incorrect query result is possible! Here we have the strange situation that a seemingly semantically correct SQL-query produces a result of which the correctness depends on the actual contents of the database!

In the semantic approach NULL-values do not occur in the database. What would be the query result of counting Amsterdam-employees per department if we allow that the value of the attribute employee its department can be NULL? Moreover, allowing NULL-values would lead to many problems because in many queries we have to walk through data paths existing in the underlying data model. The above mentioned extend operation applies a path consisting of only one attribute: employee its department. However, longer paths may be used as in the following example where the path employee its department its town is involved in a query about commuters:

- *get* employee *its* name, address, department, town, department *its* town
- *where* not town = department *its* town.

In many cases users want to retrieve information about data sets, but it is also possible and simple to get information about some specific instance, for example an employee with the identification 233:

- *get* employee 233 *its* name, address, town, department.

3.2 Organisation of Semantic Data Models

The piece of the "real-world" represented by a semantic model is referred to as an enterprise. A semantic schema represents important elements within an enterprise and shows the structural interrelationships among those elements. Virtually all semantic data models offer a diagrammatic

construct for conceptualising schemas. The fundamental components used by semantic models to structure data are objects, atomic and constructed types, attributes, ISA relationships, and derived schema components. We begin by discussing objects, atomic types, constructed types and so on.

1. **Objects** − Elements within an enterprise are represented by objects or entities. It is generally assumed that these objects can be any unstructured data such as strings, integers, and reals, or, in the case of multimedia enterprises, text, voice and image data.

2. **Atomic types** − The direct representation of object types, distinct from their attributes, is essential to semantic modelling. As the name implies, atomic types correspond to classes of simple, non- aggregate objects in an enterprise. The Hypertext Community schema in Figure 1 shows the organisation of object types in a hypothetical enterprise. In it, two atomic types, PERSON and EMPLOYER, are represented. These are depicted using triangles to indicate that they are abstract or nonprintable types.

   Abstract types generally correspond directly to physical or conceptual objects in an enterprise. Depending upon a particular model's implementation, these types may not be visible to the user. In contrast, printable types, such as PNAME and OCCUPATION, would have an external representation accessible to the user. The active domain of an atomic type holds all objects of that type currently in the database.

3. **Constructed types** − Perhaps the most important feature of semantic models is their ability to construct complex object types from atomic types. Aggregation and grouping, also called association, are the most common type constructors in the semantic literature.

   An aggregation is a composite type formed from other types already existing in the schema. Mathematically, an aggregation is an ordered n-tuple, and, in an instance of the schema the active domain of an aggregation type will be a subset of the Cartesian product of the active domains assigned to its underlying nodes. The aggregation abstraction allows users to focus on a complex data type while ignoring its component parts. The grouping constructor, on the other hand, is used to represent sets of objects of the same type.

   Mathematically, a grouping is a finitary power set. In an instance, the active domain of a grouping type will consist of a set of objects, each of which is a finite subset of the active domain of the underlying node. Although aggregation and grouping define new object types from previously defined types, they are fundamentally distinct abstractions. Aggregation, in effect, provides a means for specifying the attributes of a new object type (see below), whereas grouping defines a type whose value will be a set of objects of a particular type. In most semantic models supporting aggregation and grouping, there can

be no directed or undirected cycles of type constructor edges.

4. **Attributes**−Another fundamental aspect of semantic models is their ability to represent interrelational dependencies or connections among object types. These properties are called attributes or relationships. Attributes are generally specified explicitly by the user to correspond to facts about an enterprise. In the Hypertext Community schema, PERSON has three attributes: HAS-NAME, LIVES-AT, and USES.

   In this schema, attributes are represented by arrows that originate at the domain of each attribute and terminate at its range. That is to say, the HAS-NAME attribute maps an object of type PERSON to a printable object of type PNAME with similar mappings implied by the other attributes. A range object can be referred to as the value of the mapping attribute. In a more formal sense, there are several ways attributes can be expressed in semantic models. The HAS-NAME attribute is an example of a one-argument attribute, i.e. it is a directed, binary relationship between two types. Some models allow n-argument attributes, which define a directed relationship between a set of n types and one type.

   Attributes can also be either single-valued or multi-valued. An ADDRESS, for example, can be the residence of several PERSONS. In our schema, single- and multi-valued relationships are distinguished by one- and two-headed arrows respectively. In an instance of the schema, a mapping is assigned to each attribute. The mapping will consist of a binary or (n+1)-ary relation, depending on the number of arguments n in the attribute. The domain of the mapping is the Cartesian product of the active domain(s) of the attribute's source(s), and the range is the active domain of the attribute's target. In the case of single-valued attributes, such as HAS-NAME, the mapping must be a function in the strict mathematical sense.

   The distinctions between various forms of type constructors and attributes help explain some of the variability one sees in the expressiveness of semantic models. For example, there is a close correspondence between the semantics of a multi-valued attribute and the semantics of a single-valued attribute whose range is a constructed grouping type. Likewise, a one-argument attribute whose domain is an aggregation and an n-argument attribute are similar.

   It should be clear that several ways of representing essentially the same data interrelationships can exist given a full range of modelling functionality. Most existing models, however, only support multi-valued attributes and do not permit an attribute to map to a grouping type. What may not be quite so obvious is that the semantics of object identity are strongly influenced by the abstractions chosen in a representation.

There is a final point about attributes. Some semantic models draw a fine distinction between attributes and relationships. In these cases, attributes are considered relationships in which the values are atomic. In addition, some models allow relationships, but not attributes, to have attributes of their own, just like other objects. In the remainder of this lecture, we use the two terms interchangeably.

5. **ISA relationships** − Virtually all semantic models have the ability to represent ISA or super-type/subtype relationships. Generally speaking, an ISA relationship from a subtype to a super-type indicates that each object associated with the subtype is also associated with the super-type.
   In most semantic models, subtypes inherit their supertype's attributes and can have attributes that are not shared by their super-type. Since semantic models usually allow undirected or weak cycles to exist among these relationships, they form what is often referred to as the schema's ISA network.

   ISA relationships are used in semantic models for two closely related purposes. First, they can represent one or more overlapping subtypes of a type. Second, they can be used to form types that contain a union of disjoint types already present in the schema. Historically, semantic models have used a single kind of ISA relationship for both purposes.
   Recent research, however, has differentiated several kinds of ISA relationships, such as subset and generalisation that allow the subtle semantics of set containment and partitioning to be expressed.

6. **Derived schema components** − Derived schema components, also called derived data, are a basic mechanism for data abstraction and encapsulation in many semantic models. Derivation allows information to be incorporated into a database schema that is itself computed from other information in the schema. Derived schema components consist of a structural specification for holding the derived information and a derivation rule describing how the structure is to be filled. The structure, in most cases, is either a derived subtype or a derived attribute.

   Generally speaking, derived data are automatically updated as required by updates to other parts of the schema. Objects, atomic and constructed types, attributes, ISA relationships, and derived schema components are the fundamental abstractions offered by the semantic modelling paradigm. It is important to note that much of the expressive power of semantic models comes from the fact that these constructs can usually be used recursively.

From a broader perspective, though, the effectiveness of semantic modelling is influenced by

several other important factors. These include such things as combining restrictions, static integrity constraints, and the manipulation languages that enable users to interact with data.

1. **Combining restrictions** − Most semantic models do not allow arbitrary combinations of the basic constructs. The most prominent of these restrictions affect the combining of ISA relationships. For example, directed cycles of ISA edges really make no sense and are usually not permitted. Generally speaking, combining restrictions assure that schemas are not redundant or ambiguous and capture the natural intuition about objects.
2. **Static integrity constraints** − In general, semantic models express in a structural manner the most important types of relational integrity constraints. There are, however, a variety of relationships and properties of relationships that cannot be directly represented with the abstractions presented thus far. If objects are connected through relationships, then insertion, deletion, or modification of one object can still impact others. For this reason, many semantic models provide mechanisms for specifying integrity constraints, which in essence, assure that data associated with different parts of a schema are consistent according to some criteria. We will return to this issue in our discussion section.
3. **Manipulation languages** − The data structuring capabilities discussed so far would normally be supported by a data definition language associated with a specific model. In addition, the model would be expected to have a corresponding data manipulation language that allows users to interact with the database. There are three fundamental capabilities that differentiate a semantic data manipulation language from a manipulation language for a traditional, record-oriented model: its ability to query abstract types, manipulate attributes, and manage derived data.

3.2.1 Semantic Data in the Database

There is one universe for all semantic data stored in the database. All triples are parsed and stored in the system as entries in tables under the MDSYS schema. A triple {subject, property, and object} is treated as one database object. As a result, a single document containing multiple triples results in multiple database objects.

All the subjects and objects of triples are mapped to nodes in a semantic data network, and properties are mapped to network links that have their start node and end node as subject and object, respectively. The possible node types are blank nodes, URIs, plain literals, and typed literals.

The following requirements apply to the specifications of URIs and the storage of semantic data in the database:

- A subject must be a URI or a blank node.
- A property must be a URI.

An object can be any type, such as a URI, a blank node, or a literal. (However, null values and null strings are not supported.)

3.2.2 Metadata for Models

The MDSYS.SEM_MODELS view contains information about all models defined in the database. When you create a model using the SEM_APIS.CREATE_SEM_MODEL procedure, you specify a name for the model, as well as a table and column to hold references to the semantic data, and the system automatically generates a model ID.

Oracle maintains the MDSYS.SEM_MODELS view automatically when you create and drop models. Users should never modify this view directly. For example, do not use SQL INSERT, UPDATE, or DELETE statements with this view.

The MDSYS.SEM_MODELS view contains the columns shown in Table 3

Table 3: MDSYS.SEM_MODELS View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Schema of the owner of the model.
MODEL_ID	NUMBER	Unique model ID number automatically generated.
MODEL_NAME	VARCHAR2(25)	Name of the model.
TABLE_NAME	VARCHAR2(30)	Name of the table to hold references to semantic data for the model.
COLUMN_NAME	VARCHAR2(30)	Name of the column of type SDO_RDF_TRIPLE_S in the table to hold references to semantic data for the model.
MODEL_TABLESPACE_NAME	VARCHAR2(30)	Name of the tablespace to be used for storing the triples for this model.

When you create a model, a view for the triples associated with the model is also created under the MDSYS schema. This view has a name in the format RDFM_model-name, and it is visible only to the owner of the model and to users with suitable privileges. Each MDSYS.SEMM_model-name view contains a row for each triple (stored as a link in a network), and it has the columns shown in Table 4.

Table 4: MDSYS.SEMM_Model-Name View Columns

Column Name	Data Type	Description
P_VALUE_ID	NUMBER	The VALUE_ID for the text value of the predicate of the triple. Part of the primary key.
START_NODE_ID	NUMBER	The VALUE_ID for the text value of the subject of the triple. Also part of the primary key.
CANON_END_NODE_ID	NUMBER	The VALUE_ID for the text value of the canonical form of the object of the triple. Also part of the primary key.
END_NODE_ID	NUMBER	The VALUE_ID for the text value of the object of the triple
MODEL_ID	NUMBER	The ID for the RDF graph to which the triple belongs. It logically partitions the table by RDF graphs.
COST	NUMBER	(Reserved for future use)
CTXT1	NUMBER	(Reserved for future use)
CTXT2	VARCHAR2 (4000)	(Reserved for future use)
DISTANCE	NUMBER	(Reserved for future use)
EXPLAIN	VARCHAR2 (4000)	(Reserved for future use)
PATH	VARCHAR2 (4000)	(Reserved for future use)
LINK_ID	VARCHAR2 (71)	Unique triple identifier value. (It is currently a computed column, and its definition may change in a future release.)

Note:

In Table 4, for columns P_VALUE_ID, START_NODE_ID, END_NODE_ID, and

CANON_END_NODE_ID, the actual ID values are computed from the corresponding lexical values. However, a lexical value may not always map to the same ID value.

3.2.3 Statements

The MDSYS.RDF_VALUE$ table contains information about the subjects, properties, and objects used to represent RDF (Resource Description Framework) statements. It uniquely stores the text values (URIs or literals) for these three pieces of information, using a separate row for each part of each triple.

Oracle maintains the MDSYS.RDF_VALUE$ table automatically. Users should never modify this view directly. For example, do not use SQL INSERT, UPDATE, or DELETE statements with this view. The RDF_VALUE$ table contains the columns shown in Table 5.

Table 5: MDSYS.RDF_VALUE$ Table Columns

Column Name	Data Type	Description
VALUE_ID	NUMBER	Unique value ID number automatically generated.
VALUE_TYPE	VARCHAR2(10)	The type of text information stored in the VALUE_NAME column. Possible values: UR for URI, BN for blank node, PL for plain literal, PL@ for plain literal with a language tag, PLL for plain long literal, PLL@ for plain long literal with a language tag, TL for typed literal, or TLL for typed long literal. A long literal is a literal with more than 4000 bytes.

VNAME_PREFIX	VARCHAR2(4000)	If the length of the lexical value is 4000 bytes or less, this column stores a prefix of a portion of the lexical value. The SEM_APIS.VALUE_NAME_PREFIX function can be used for prefix computation. For example, the prefix for the portion of the lexical value <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> without the angle brackets is http://www.w3.org/1999/02/22-rdf-syntax-ns#.
VNAME_SUFFIX	VARCHAR2(512)	If the length of the lexical value is 4000 bytes or less, this column stores a suffix of a portion of the lexical value. The SEM_APIS.VALUE_NAME_SUFFIX function can be used for suffix computation. For the lexical value mentioned in the description of the VNAME_PREFIX column, the suffix is type.
LITERAL_TYPE	VARCHAR2(4000)	For typed literals, the type information; otherwise, null. For example, for a row representing a creation date of 1999-08-16, the VALUE_TYPE column can contain TL, and the LITERAL_TYPE column can contain http://www.w3.org/2001/XMLSchema#date.
LANGUAGE_TYPE	VARCHAR2(80)	Language tag (for example, fr for French) for a literal with a language tag (that is, if VALUE_TYPE is PL@ or PLL@). Otherwise, this column

Column Name	Data Type	Description
		has a null value.
CANON_ID	NUMBER	The ID for the canonical lexical value for the current lexical value. (The use of this column may change in a future release.)
COLLISION_EXT	VARCHAR2(64)	Used for collision handling for the lexical value. (The use of this column may change in a future release.)
CANON_COLLISION_EXT	VARCHAR2(64)	Used for collision handling for the canonical lexical value. (The use of this column may change in a future release.)
LONG_VALUE	CLOB	The character string if the length of the lexical value is greater than 4000 bytes. Otherwise, this column has a null value.
VALUE_NAME	VARCHAR2(4000)	This is a computed column. If length of the lexical value is 4000 bytes or less, the value of this column is the concatenation of the values of VNAME_PREFIX column and the VNAME_SUFFIX column.

Oracle maintains the MDSYS.RDF_VALUE$ table automatically. Users should never modify this view directly. For example, do not use SQL INSERT, UPDATE, or DELETE statements with this view. The RDF_VALUE$ table contains the columns shown in Table 5.

3.2.4 Triple Uniqueness and Data Types for Literals

Duplicate triples are not stored in the database. To check if a triple is a duplicate of an existing triple, the subject, property, and object of the incoming triple are checked against triple values in the specified model. If the incoming subject, property, and object are all URIs, an exact match of their values determines a duplicate. However, if the object of incoming triple is a literal, an exact match of the subject and property, and a value (canonical) match of the object, determine a duplicate. For example, the following two triples are duplicates:

- <eg:a> <eg:b> "123"^^http://www.w3.org/2001/XMLSchema#int

- <eg:a> <eg:b>
- "123"^^http://www.w3.org/2001/XMLSchema#unsignedByte

The second triple is treated as a duplicate of the first, because

- "123"^^http://www.w3.org/2001/XMLSchema#int has an equivalent value (is canonically equivalent) to "123"^^http://www.w3.org/2001/XMLSchema#unsignedByte.

Two entities are canonically equivalent if they can be reduced to the same value.
To use a non-RDF example, A*(B-C), A*B-C*A, (B-C)*A, and - A*C+A*B all convert into the same canonical form.
Value-based matching of lexical forms is supported for the following data types:

- **STRING:** plain literal, xsd:string and some of its XML Schema subtypes
- **NUMERIC:** xsd:decimal and its XML Schema subtypes, xsd:float, and xsd:double. (Support is not provided for float/double INF, -INF, and NaN values.)
- **DATETIME:** xsd:datetime, with support for time zone. (Without time zone there are still multiple representations for a single value, for example, "2004-02-18T15:12:54" and "2004-02- 18T15:12:54.0000".)
- **DATE:** xsd:date, with or without time zone
- **OTHER:** Everything else. (No attempt is made to match different representations).
- Canonicalisation is performed when the time zone is present for literals of type xsd:time and xsd:dateTime.

The following namespacedefinition is used: xmlns:xsd=”http://www.w3.org/2001/XMLSchema”
The first occurrence of a literal in the RDF_VALUES table is taken as the canonical form and given the VALUE_TYPE value of CPL, CPL@, CTL, CPLL, CPLL@, or CTLL as appropriate; that is, a C for canonical is prefixed to the actual value type. If a literal with the same canonical form (but a different lexical representation) as a previously inserted literal is inserted into the RDF_VALUES table, the VALUE_TYPE value assigned to the new insert is PL, PL@, TL, PLL, PLL@, or TLL as appropriate.
Canonically equivalent text values having different lexical representations are thus stored in the RDF_VALUES table; however, canonically equivalent triples are not stored in the database.

3.2.5 Subjects and Objects

RDF subjects and objects are mapped to nodes in a semantic data network. Subject nodes are the start nodes of links, and object nodes are the end nodes of links. Non-literal nodes (that is, URIs and blank nodes) can be used as both subject and object nodes. Literals can be used only as object nodes.

3.2.6 Blank Nodes

Blank nodes can be used as subject and object nodes in the semantic network. Blank node

identifiers are different from URIs in that they are scoped within a semantic model. Thus, although multiple occurrences of the same blank node identifier within a single semantic model necessarily refer to the same resource, occurrences of the same blank node identifier in two different semantic models do not refer to the same resource. In an oracle semantic network, this behaviour is modelled by requiring that blank nodes are always reused (that is, are used to represent the same resource if the same blank node identifier is used) within a semantic model, and never reused between two different models. Thus, when inserting triples involving blank nodes into a model, you must use the SDO (Service Data Objects)_RDF_TRIPLE_S constructor that supports reuse of blank nodes.

3.2.7 Properties

Properties are mapped to links that have their start node and end node as subjects and objects, respectively. Therefore, a link represents a complete triple. When a triple is inserted into a model, the subject, property, and object text values are checked to see if they already exist in the database. If they already exist (due to previous statements in other models), no new entries are made; if they do not exist, three new rows are inserted into the RDF_VALUES table.

3.2.8 Inferencing: Rules and Rulebases

Inferencing is the ability to make logical deductions based on rules. Inferencing enables you to construct queries that perform semantic matching based on meaningful relationships among pieces of data, as opposed to just syntactic matching based on string or other values. Inferencing involves the use of rules, supplied by oracle or user-defined, placed in rulebases.

Figure 25 shows triple sets being inferred from model data and the application of rules in one or more rulebases. In this illustration, the database can have any number of semantic models, rulebases, and inferred triple sets, and an inferred triple set can be derived using rules in one or more rulebases.

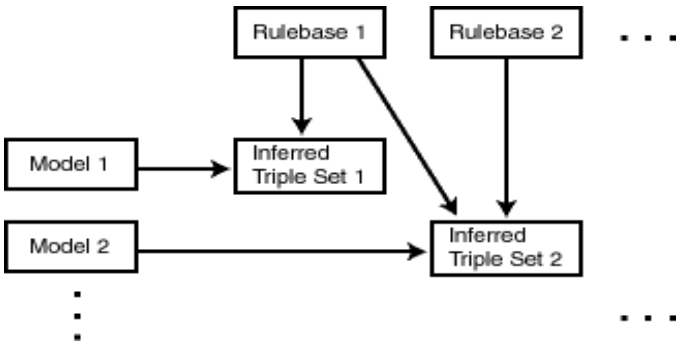


Fig. 34: Inferencing

Description of "Figure 34 Inferencing"

A rule is an object that can be applied to draw inferences from semantic data. A rule is identified

by a name and consists of:

- An IF side pattern for the antecedents
- An optional filter condition that further restricts the subgraphs matched by the IF side pattern
- A THEN side pattern for the consequents

For example, the rule that a chairperson of a conference is also a reviewer of the conference could be represented as follows:

```
('chairpersonRule', -- rule name
'(?r :ChairPersonOf ?c)', -- IF side pattern
 NULL, -- filter condition
'(?r :ReviewerOf ?c)', -- THEN side pattern
SEM_ALIASES (SEM_ALIAS ('',
'http://some.org/test/')))
```

In this case, the rule does not have a filter condition, so that component of the representation is NULL. Note that a THEN side pattern with more than one triple can be used to infer multiple triples for each IF side match.
A rulebase is an object that contains rules. The following oracle- supplied rulebases are provided:

- RDFS
- RDF (a subset of RDFS)
- OWLSIF (empty)
- RDFS++ (empty)
- OWLPRIME (empty)

The RDFS and RDF rulebases are created when you call the SEM_APIS.CREATE_SEM_NETWORK procedure to add RDF support to the database. The RDFS rulebase implements the RDFS entailment rules, as described in the World Wide Web Consortium (W3C) RDF Semantics document at http://www.w3.org/TR/rdf-mt/. The RDF rulebase represents the RDF entailment rules, which are a subset of the RDFS entailment rules. You can see the contents of these rulebases by examining the MDSYS.SEMR_RDFS and MDSYS.SEMR_RDF views.

You can also create user-defined rulebases using the SEM_APIS.CREATE_RULEBASE procedure. User-defined rulebases enable you to provide additional specialised inferencing capabilities.
For each rulebase, a system table is created to hold rules in the rulebase, along with a system view with a name in the format MDSYS.SEMR_rulebase-name (for example,

MDSYS.SEMR_FAMILY_RB for a rulebase named FAMILY_RB). You must use this view to insert, delete, and modify rules in the rulebase. Each MDSYS.SEMR_rulebase-name view has the columns shown in Table 6 below.

Table 6: MDSYS.SEMR_Rulebase-Name View Columns

Column Name	Data Type	Description
RULE_NAME	VARCHAR2(30)	Name of the rule
ANTECEDENTS	VARCHAR2(4000)	IF side pattern for the antecedents
FILTER	VARCHAR2(4000)	Filter condition that further restricts the subgraphs matched by the IF side pattern. Null indicates no filter condition is to be applied.
CONSEQUENTS	VARCHAR2(4000)	THEN side pattern for the consequents
ALIASES	SEM_ALIASES	One or more namespaces to be used. (The SEM_ALIASES data type is described in Section 1.6.)

Information about all rulebases is maintained in the MDSYS.SEM_RULEBASE_INFO view, which has the columns shown in Table 7 and one row for each rulebase.

Table 7: MDSYS.SEM_RULEBASE_INFO View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Owner of the rulebase
RULEBASE_NAME	VARCHAR2(25)	Name of the rulebase
RULEBASE_VIEW_NAME	VARCHAR2(30)	Name of the view that you must use for any SQL statements that insert, delete, or modify rules in the rulebase

STATUS	VARCHAR2(30)	Contains VALID if the rulebase is valid, INPROGRESS if the rulebase is being created, or FAILED if a system failure occurred during the creation of the rulebase.

For example, the rule that a chairperson of a conference is also a reviewer of the conference could be represented as follows:

Example 1-1: creates a rulebase named family_rb, and then inserts a rule named grandparent_rule into the family_rb rulebase. This rule says that if a person is the parent of a child who is the parent of a child, that person is a grandparent of (that is, has the grandParentOf relationship with respect to) his or her child's child. It also specifies a namespace to be used.

Example 1-1 Inserting a Rule into a Rulebase

```
EXECUTE SEM_APIS.CREATE_RULEBASE('family_rb');
NSERT INTO mdsys.semr_family_rb
VALUES('grandparent_rule', '(?x :parentOf ?y) (?y :parentOf
?z)',
NULL,
'(?x :grandParentOf ?z)',
```

SEM_ALIASES (SEM_ALIAS('','http://www.example.org/family/'))); You can specify one or more rulebases when calling the SEM_MATCH table function (described in Section 1.6), to control the behaviour of queries against semantic data. Example 1-2 refers to the family_rb rulebase and to the grandParentOf relationship created in Example 1-1, to find all grandfathers (grandparents who are male) and their grandchildren.

Example 1-2 Using Rulebases for Inferencing

```
-- Select all grandfathers and their grandchildren from the family model.
-- Use inferencing from both the RDFS and family_rb rulebases.
SELECT x, y
  FROM TABLE (SEM_MATCH(
    '(?x :grandParentOf ?y) (?x rdf:type :Male)',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/'
    )), null));
```

3.2.9 Rules Indexes

A rules index is an object containing precomputed triples that can be inferred from applying a specified set of rulebases to a specified set of models. If a SEM_MATCH query refers to any rulebases, a rules index must exist for each rulebase-model combination in the query.

To create a rules index, use the SEM_APIS.CREATE_RULES_INDEX procedure. To drop (delete) a rules index, use the SEM_APIS.DROP_RULES_INDEX procedure.

When you create a rules index, a view for the triples associated with the rules index is also created under the MDSYS schema. This view has a name in the format SEMI_rules-index-name, and it is visible only to the owner of the rules index and to users with suitable privileges. Each MDSYS.SEMI_rules-index-name view contains a row for each triple (stored as a link in a network), and it has the same columns as the SEMM_model-name view, which is described in Table 4 above. Information about all rules indexes is maintained in the MDSYS.SEM_RULES_INDEX_INFO view, which has the columns shown in Table 8 and one row for each rules index.

Table 8: MDSYS.SEM_RULES_INDEX_INFO View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Owner of the rules index
INDEX_NAME	VARCHAR2(25)	Name of the rules index
INDEX_VIEW_NAME	VARCHAR2(30)	Name of the view that you must use for any SQL statements that insert, delete, or modify rules in the rules index
STATUS	VARCHAR2(30)	Contains VALID if the rules index is valid, INVALID if the rules index is not valid, INCOMPLETE if the rules index is incomplete (similar to INVALID but requiring less time to re-create), INPROGRESS if the rules index is being created, or FAILED if a system failure occurred during the creation of the rules index.

MODEL_COUNT	NUMBER	Number of models included in the rules index
RULEBASE_COUNT	NUMBER	Number of rulebases included in the rules index

Information about all database objects, such as models and rulebases, related to rules indexes is maintained in the MDSYS.SEM_RULES_INDEX_DATASETS view. This view has the columns shown in Table 9 and one row for each unique combination of values of all the columns.

Table 9: MDSYS.SEM_RULES_INDEX_DATASETS View Columns

Column Name	Data Type	Description
INDEX_NAME	VARCHAR2(25)	Name of the rules index
DATA_TYPE	VARCHAR2(8)	Type of data included in the rules index. Examples: MODEL and RULEBASE
DATA_NAME	VARCHAR2(25)	Name of the object of the type in the DATA_TYPE column

Example 1-3: creates a rules index named family_rb_rix_family, using the family model and the RDFS and family_rb rulebases.

Example 1-3 Creating a Rules Index

```
BEGIN
SEM_APIS.CREATE_RULES_IND
EX(
'rdfs_rix_family',
SEM_Models('family'),
SEM_Rulebases('RDFS','family_rb'
)); END;
/
```

3.2.10 Virtual Models

A virtual model is a logical graph that can be used in a SEM_MATCH query. A virtual model is the result of a UNION or UNION ALL operation on one or more models and optionally the corresponding rules index.

Using a virtual model can simplify management of access privileges for semantic data. For example, assume that you have created three semantic models and one rules index based on the three models and the OWLPRIME rulebase. Without a virtual model, you must individually grant and revoke access privileges for each model and the rules index. However, if you create a virtual model that contains the three models and the rules index, you will only need to grant and

revoke access privileges for the single virtual model.

Using a virtual model can also facilitate rapid updates to semantic models. For example, assume that virtual model VM1 contains model M1 and rules index R1 (that is, VM1 = M1 UNION ALL R1), and assume that semantic model M1_UPD is a copy of M1 that has been updated with additional triples and that R1_UPD is a rules index created for M1_UPD. Now, to have user queries over VM1 go to the updated model and rules index, you can redefine virtual model VM1 (that is, VM1 = M1_UPD UNION ALL R1_UPD).
To create a virtual model, use the SEM_APIS.CREATE_VIRTUAL_MODEL procedure. To drop (delete a virtual model, use the SEM_APIS.DROP_VIRTUAL_MODEL procedure. A virtual model is dropped automatically if any of its component models, rulebases, or rules index are dropped.

To query a virtual model, specify the virtual model name in the models parameter of the SEM_MATCH table function, as shown in Example 1- 4.

Example 1-4 Querying a Virtual Model

```
SELECT COUNT(protein)
  FROM TABLE (SEM_MATCH )
    '(?protein rdf:type :Protein)
    (?protein :citation ?citation)
    (?citation :author "Bairoch
    A.")',
    SEM_MODELS('UNIPROT_V
    M'), NULL,
    SEM_ALIASES(SEM_ALIAS('', 'http://purl.uniprot.org/core/')),
    NULL,
    NULL,
    'ALLOW_DUP=T'));
```

When you create a virtual model, an entry is created for it in the MDSYS.SEM_MODEL$ view, which is described in Table 3 above. However, the values in several of the columns are different for virtual models as opposed to semantic models, as explained in Table 10.

Table 10: MDSYS.SEM_MODEL$ View Column Explanations for Virtual Models

Column Name	Data Type	Description
OWNER	VARCHAR2 (30)	Schema of the owner of the virtual model.

MODEL_ID	NUMBER	Unique model ID number automatically generated. Will be a negative number, to indicate that this is a virtual model.
MODEL_NAME	VARCHAR2 (25)	Name of the virtual model.
TABLE_NAME	VARCHAR2 (30)	Null for a virtual model.
COLUMN_NAME	VARCHAR2 (30)	Null for a virtual model.
MODEL_TABLESPACE_NAME	VARCHAR2 (30)	Null for a virtual model.

Information about all virtual models is maintained in the MDSYS.SEM_VMODEL_INFO view, which has the columns shown in Table and one row for each virtual model.

Table 11: MDSYS.SEM_VMODEL_INFO View Columns

Column Name	**Data Type**	**Description**
OWNER	VARCHAR2(30)	Owner of the virtual model
VIRTUAL_MODEL_NAME	VARCHAR2(25)	Name of the virtual model
UNIQUE_VIEW_NAME	VARCHAR2(30)	Name of the view that contains unique triples in the virtual model, or null if the view was not created
DUPLICATE_VIEW_NAME	VARCHAR2(30)	Name of the view that contains duplicate triples (if any) in the virtual model

TATUS	VARCHAR2(30)	Contains VALID if the associated rules index is valid, INVALID if the rules index is not valid, INCOMPLETE if the rules index is incomplete (similar to INVALID but requiring less time to re-create), INPROGRESS if the rules index is being created, FAILED if a system failure occurred during the creation of the rules index, or NORIDX if no rules index is associated with the virtual model.
MODEL_COUNT	NUMBER	Number of models in the virtual model
RULEBASE_COUNT	NUMBER	Number of rulebases used for the virtual model
RULES_INDEX_COUNT	NUMBER	Number of rules indexes in the virtual model

Information about all objects (models, rulebases, and rules index) related to virtual models is maintained in the MDSYS.SEM_VMODEL_DATASETS view. This view has the columns shown in Table 11 and one row for each unique combination of values of all the columns.

Table 12: MDSYS.SEM_VMODEL_DATASETS View Columns

Column Name	**Data Type**	**Description**
VIRTUAL_MODEL_NAME	VARCHAR2(25)	Name of the virtual model
DATA_TYPE	VARCHAR2(8)	Type of object included in the virtual model. Examples: MODEL for a semantic model, RULEBASE for a rulebase, or RULEIDX for a rules index

DATA_NAME	VARCHAR2(25)	Name of the object of the type in the DATA_TYPE column

3.2.11 Semantic Data Security Considerations

The following database security considerations apply to the use of semantic data:

- **a.** When a model or rules index is created, the owner gets the SELECT privilege with the GRANT option on the associated view. Users that have the SELECT privilege on these views can perform SEM_MATCH queries against the associated model or rules index.
- **b.** When a rulebase is created, the owner gets the SELECT, INSERT, UPDATE, and DELETE privileges on the rulebase, with the GRANT option. Users that have the SELECT privilege on a rulebase can create a rules index that includes the rulebase. The INSERT, UPDATE, and DELETE privileges control which users can modify the rulebase and how they can modify it.
- **c.** To perform data manipulation language (DML) operations on a model, a user must have DML privileges for the corresponding base table.
- **d.** The creator of the base table corresponding to a model can grant privileges to other users.
- **e.** To perform data manipulation language (DML) operations on a rulebase, a user must have the appropriate privileges on the corresponding database view.
- **f.** The creator of a model can grant SELECT privileges on the corresponding database view to other users.
- **g.** A user can query only those models for which that user has SELECT privileges to the corresponding database views.
- **h.** Only the creator of a model or a rule base can drop it.

3.3 Semantic Annotation, Indexing, and Retrieval

Annotation, or tagging, is about attaching names, attributes, comments, descriptions, etc. to a document or to a selected part in a text. It provides additional information (metadata) about an existing piece of data. The semantic annotation is a specific metadata generation and usage schema targeted to enable new information access methods and extend existing ones. The annotation scheme is based on the understanding that the named entities mentioned in the documents constitute important part of their semantics.

In a nutshell, Semantic Annotation is about assigning the entities in the text links to their semantic descriptions (as presented on Figure 1 below). This sort of metadata provides both class and instance information about the entities. It is a matter of terminology whether these annotations should be called “semantic”, “entity” or some other way.

Semantic Annotation helps to bridge the ambiguity of the natural language when expressing notions and their computational representation in a formal language. By telling a computer how

data items are related and how these relations can be evaluated automatically, it becomes possible to process complex filter and search operations.

3.3.1 Semantic Annotation Model and Representation

Here we will discuss the structure and the representation of the semantic annotations, including the necessary knowledge and metadata. There are number of basic prerequisite for representation of semantic annotations:

- **Ontology** (or at least taxonomy) defining the entity classes. It should be possible to refer to those classes;
- **Entity identifiers** which allow those to be distinguished and linked to their semantic descriptions;
- Knowledge base with entity descriptions.

The next question considers an important choice for the representation of the annotations – "to embed or not to embed?" Although the embedded annotations seem easier to maintain, there are number of arguments providing evidence that the semantic annotations have to be decoupled from the content they refer to. One key reason is to allow dynamic, user-specific, semantic annotations – the embedded annotations become part of the content and may not change corresponding to the interest of the user or the context of usage.

Further, embedded complex annotations would have negative impact on the volume of the content and can complicate its maintenance – imagine that page with three layers of overlapping semantic annotations need to be updated preserving them consistent. Once decided that the semantic annotations has to be kept separate from the content, the next question is whether or not (or how much) to couple the annotations with the ontology and the knowledge base? It is in this case that such integration seems profitable – it would be easier to keep in synch the annotations with the class and entity descriptions. However, there are at least two important problems:

Both the cardinality and the complexity of the annotations differ from those of the entity descriptions – the annotations are simpler, but their count is usually much bigger than this of the entity descriptions. Even considering middle-sized document corpora the annotations can reach tens of millions. Suppose 10M annotations are stored in an RDF(S) store together with 1M entity descriptions. Suppose also that each annotation and each entity description are represented with 10 statements. There is a considerable difference regarding the inference approaches and hardware capable in efficient reasoning and access to 10Mstatement repository and with 110M statement repository.

It would be nice if the world knowledge (ontology and instance data) and the document-related metadata are kept independent. This would mean that for one and the same document different extraction, processing, or authoring methods will be able to deliver alternative metadata referring

to one and the same knowledge store.
Most important, it should be possible for the ownership and the responsibility for the metadata and the knowledge to be distributed. This way, different parties can develop and maintain separately the content, the metadata, and the knowledge.
Based on the above arguments, it is proposed to decouple representation and management of the documents, the metadata (annotations) and the formal knowledge (ontologies and instance data).

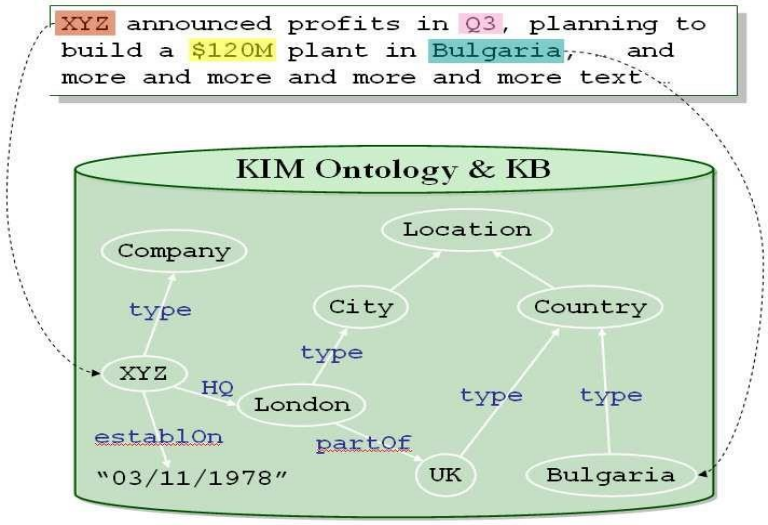


Fig. 35: Semantic Annotation

3.3.2 Semantic Annotation Process

As already mentioned, we focus mainly on the automatic semantic annotation, leaving manual annotation to approaches more related to authoring web content. Even less accurate, the automatic approaches for metadata acquisition promise scalability and without them the Semantic Web will remain mostly a vision for long time. Our experience shows that the existing state-of-the-art IE systems have the potential to automate the annotation with reasonable accuracy and performance.

Although a lot of research and development contributed in the area of automatic IE so far, the lack of standards and integration with formal knowledge management systems were obscuring its usage. We claim that it is crucial to encode the extracted knowledge formally and according to well known and widely accepted knowledge representation and metadata encoding standards. Such system should be easily extensible for domain-specific applications, providing basic means for addressing the most common entities types, their attributes, and relations.

3.3.3 Indexing and Retrieval

Historically, the issue of specific handling of the named entities was neglected by the information retrieval (IR) community, apart from some shallow handling for the purpose of Questions/Answering tasks. However, a recent large scale human interaction study on a personal

content IR system of Microsoft demonstrates that, at least in some cases, the ignorance of the named entities does not match the user needs. And based on semantic annotations, efficient indexing and retrieval techniques could be developed involving explicit handling of the named entity references.
In a nutshell, the semantic annotations could be used to index both "NY" and "N.Y." as occurrence of the specific entity "New York" like if there was just its unique ID. Because of no entity recognition involved, the present systems will index on "NY", "N", and "Y" which demonstrates well some of the problems with the keyword-based search engines. Given metadata indexing of the content, advanced semantic querying should be feasible.

In a query towards a repository of semantically annotated documents, it should be possible to specify entity type restrictions, name and other attribute restrictions, as well as relations between the entities of interest. For instance, it should be possible to make a query that targets all documents that refer to Persons that hold some Positions within an Organisation, and also restricts the names of the entities or some of their attributes (e.g. a person's gender). Further, semantic annotations could be used to match specific references in the text to more general queries. For instance, a query such as "company 'Redwood Shores' could match documents mentioning the town and specific companies such as Oracle and Symbian, but not the word "company".
Finally, although the above sketched enhancements look prominent, it still requires a lot of research and experiments to determine to what extent and how they could improve the existing IR systems. It is hard in a general context to predict how semantic indexing will combine with the symbolic and the statistical methods currently in use, such as the lexical approach presented in [20] and the latent semantic analysis presented in [18]. For this purpose, large scale experimental data and evaluation are required.

3.4 Entities Concept in Semantic Data Modelling Representing the Real World with Entities in Semantic Data Modelling

In Semantic Data Modelling (SDM), an entity represents some aspect or item in the real world, such as an employee. An entity is akin to a record in a relational system or an object in an object-oriented system. These entities in SDM focus on types, which are more general, instead of sets of data. In SDM, an entity is a very basic notion of a real-world or conceptual object that is defined by a single attribute.

For instance, an SDM entity type might be person, which provides an elementary category that can be easily understood. In a relational model, however, you might end up with a number of different tables "including person, spouse, children, house, and job." Each of these things represents part of what makes up the person, but with SDM, the person is the whole entity, rather than breaking it down into parts.

In this way, an entity in SDM is very similar to a domain. Therefore, inside this domain of person, there would be a list of names of people that are to be represented by the data. The objects in this domain would then point to specific instances of a person that are represented by each person entity. For example, the domain Person names contain Bob, Sue, Jim, Betty, and Clyde. Each of these names points to a specific object instance of Person, so that Bob points to a record giving detail about Bob, such as name, gender, or marital status, and so on for each of the entities listed under Person.

3.5 Concept of Design and Indexing

Data model designs are the first and most important piece of any database application. The first logical model is about the business requirements. After the logical model is complete and approved, the physical model is materialised into a database with constraints and indexes, data types, and so on. The data model also indicates how queries can be correctly written. Joins should be used only along the relationship lines. Because there tends to be a lack of modelling discipline in some database groups, developers join on columns that seem to have the same name or whatever criteria they can come up with.

In the plant model, although there is an Area ID in the Sensor table, there is not a relationship line, so there should not be a join between those two tables. In fact, if a query is written that way, duplicate rows will be returned and a DISTINCT clause would be required. Joining all the tables' results in good performance, joining numerous tables is not a problem; the problem is incorrect joining of numerous tables.

After the semantic data model is relationally correct, the next step is to add indexes for all foreign key constraints as shown in the model, and to add a single column index for each column in tables that you anticipate will end up in predicates. When there are many small indexes, the optimiser can put them together in the most appropriate manner.

It is a best practice to have the primary key be the clustered key in semantic models, unless there are good reasons not to (such as performance tests that reveal a problem with this). It is important to reiterate that there is an optimal balance between an object view and a relational view of the data and you can find this for your application only by testing.

4.0 Self-Assessment Exercise(s)

1. Explain the term data language and what led to its development.
2. Mention and explain the fundamental components of semantic models.
3. What do you understand by semantic data annotation?
4. Define entity within the context of semantic data modelling.

5.0 Conclusion

Semantic data models can be very complex and until semantic databases are commonly available, the challenge remains to find the optimal balance between the pure object model and the pure relational model for each application. Semantic data models attempt to provide more powerful mechanisms for structuring objects than are typically provided by traditional approaches. Relational methods, for example, emphasise information structures that promote efficient storage and retrieval rather than those that accommodate inter-object relationships.

6.0 Summary

In this unit, you have learnt that:

- the pitfalls of relational language led to the development of semantic data language
- the fundamental components used by semantic models to structure data include objects, atomic and constructed types, attributes, ISA relationships, and derived schema components
- annotation is all about attaching names, attributes, comments, descriptions, etc. to a document or to a selected part in a text
- the prerequisites for the representation of semantic data models are ontology and entity identifier
- in Semantic Data Modelling (SDM), an entity represents some aspect or item in the real world, such as an employee.

7.0 Further Readings

Dean, M.; Connolly, D.; Van, Harmelen F.; Hendler J.; Horrocks I.; McGuinness D.; Patel-Schneider P. and Stein L.A., *Web Ontology Language (OWL) Reference Version 1.0.* W3C

Dumais, S.; Cutrell, E.; Cadiz, J.; Jancke, G.; Sarin, R. & Robbins, D. *Stuff I've Seen: A System for Personal Information Retrieval and re-use. In*: proc. of SIGIR'03, July 28 –August 1, 2003, Toronto, Canada, ACM Press, pp. 72-79.

Fensel, D. *Ontology Language, v.2 (Welcome to OIL)* . Deliverable 2, On-To-Knowledge project, Dec 2001. http://www.ontoknowledge.org/downl/del2.pdf

Johan, T. Bekke (1992). *Semantic Data Modelling*. Prentice Hall. Working Draft 12 Nov. 2002, http://www.w3.org/TR/2002/WD-owl-ref-20021112/

Hills, T. (2019). *NoSQL and SQL data modeling: Bringing together data, semantics, and software*. New York: Technics publications.

Hussein, T., Paulheim, H., Lukosch, S., Ziegler, J., & Calvary, G. (2016). *Semantic models for adaptive interactive systems*. London: Springer.

Qin, Z., & Tang, Y. (2014). *Uncertainty modeling for data mining a label semantics approach*. Hangzhou: Zhejiang Univ. Press.

Tutcher, J. (2015). *Development of semantic data models to support data interoperability in the rail industry*. Birmingham: University of Birmingham.

Uzun, A. (2019). *Semantic Modeling and Enrichment of Mobile and WiFi Network Data*. Cham: Springer International Publishing.

MODULE 3: AREAS OF APPLICATION OF SEMANTIC MODELLING

Module Introduction

Introduce the module and state the units under the module.

Unit 1: Application in Computer

Unit 2: Application in Business

Unit 1: APPLICATION IN COMPUTER

Contents

1.0 Introduction

The World Wide Web started as a small Internet based hypertext project at CERN (European Organisation for Nuclear Research) in late 1990. In the past ten years the Web matured to the largest distributed hypermedia system. Now, the Web with its millions of servers, pages and users constitutes the largest library of human knowledge mankind has ever created. Hypermedia systems in general and the Web in particular may be seen as a special kind of database management systems. Thus, they commit to a certain data model, at the physical, logical and semantic level.

And despite the fact that, the Web overcame these early problems it still has a number of serious limitations. Considering the Web as the largest library of human knowledge, the Web still does not support efficiently, if at all, a number of well-known knowledge transfer processes. For instance, accessing a relevant information chunk on the Web, which constitutes so-called knowledge mining process, is a rather difficult task to accomplish by means of the current Web technologies.

2.0 Intended Learning Outcomes (ILOs)

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

- explain the origin of hypertext and hypermedia
- differentiate between Hypermedia and HC Data Models
- state and explain the classification of hypermedia systems
- explain the application of semantic Modelling to Hypermedia.

3.0 Main Content

3.1 Hypermedia System and Www

3.1.1 Overview of Logical Hypermedia Data Modelling

This chapter provides the overview of the most important logical data modelling paradigms in hypermedia systems in general and World Wide Web in particular.

3.1.2 The Definition of Hypertext and Hypermedia

As opposed to the typical printed book, where the text is read sequentially from the beginning to the end, hypertext offers a nonlinear access to the text. A typical hypertext consists of a number of chunks of textual information, usually called hyper nodes or simply nodes. Such nodes are interrelated by means of computer navigable links. Usually, links are denoted as highlighted phrases within hypertext nodes.

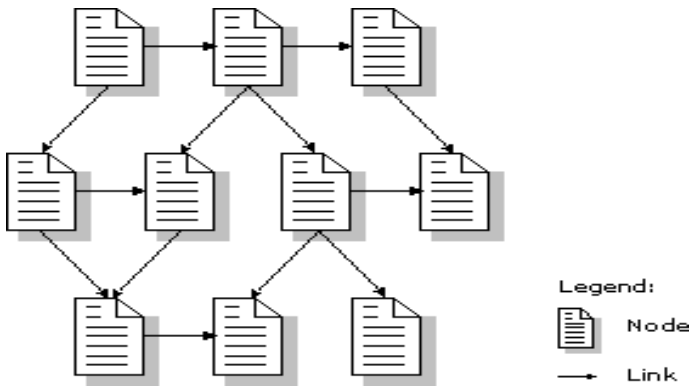


Fig. 36: Hypertext

In hypertext, readers are not confronted with a prescribed reading sequence as it is the case with a book text, but rather they browse hypertext nodes activating computer navigable links. The nature of hypertext encourages readers to explore offered information intuitively, by association, following always to their most recent interests.
Hypermedia is a generalisation of hypertext. In hypermedia nodes are multimedia nodes, i.e., they include not only textual information, but also other kinds of media, such as: digital images, sound clips, video clips and similar. In a sense, hypermedia combines hypertext and multimedia paradigm into a new one. Often, hypermedia is referred to as the concept of imposing a navigable structure on the top of existing collection of multimedia nodes [Maurer and Scherbakov, 1996; Maurer et al., 1998]. As Conklin (1987) states; hypermedia is a style of building systems for organising, structuring and accessing information around a network of multimedia nodes connected together by links.

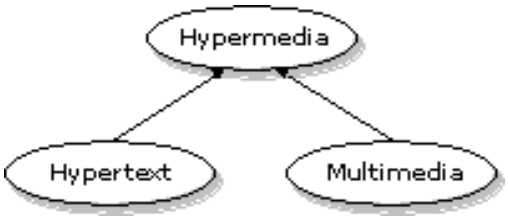


Fig. 37: Hypermedia

A source anchor is the starting point of a link and specifies a part of a node where the link may be activated. Usually, source anchors are visualised in a special way (i.e., a piece of text may be highlighted) to notify users of the existence of a link. Similarly, a destination anchor is the ending point of a link and specifies a part of a node, which should be visualised upon the link activation. Usually, an entire node is specified as the destination anchor, but a specific part of a node may be specified as the destination anchor(e.g. a paragraph within a textual node) as well.

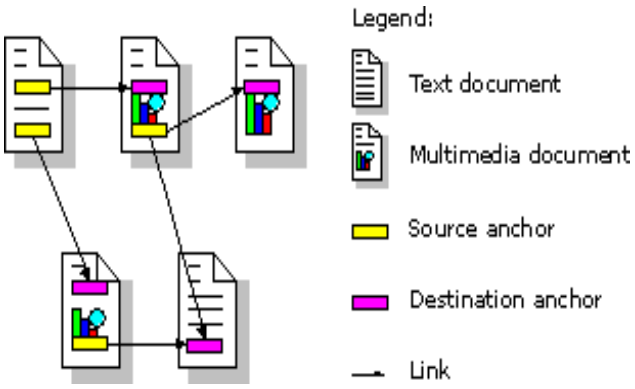


Fig. 38: Documents, Links and Anchors in Hypermedia

According to the basic structuring of data into nodes and links this basic hypermedia model is

referred to as the node-link data model. Virtually, all other hypermedia models and systems may be found in this basic hypermedia data model. Likewise, a large part of hypermedia research assumes the underlying existence of this basic model [Rivlin*etal*.,1994]. According to this model, nodes and links in hypermedia systems form a directed graph network. Normally, such a network is called hyperweb or hyperspace.

In the early days of hypermedia development a hyperweb consisted of nodes containing mostly static multimedia information. Browsing such a hyperweb meant merely retrieving simple non-interactive multimedia documents. Recently, with the development of more advanced user interaction facilities hypermedia systems usually provide users with nodes containing interactive movies, virtual reality, collaboration and communication facilities, structured discussion forums and similar.

Hypermedia systems

We can classify hypermedia systems according to a number of criteria. For instance, considering the criteria of how much control over the form of the hypermedia presentation authors should have we get the following classification:

- Frame- based hypermedia systems (all documents must fit into a fixed size frame)
- Window- based hypermedia systems (documents may be of any size and they are displayed in a scrollable window area).

There are a number of other useful classification criteria for hypermedia systems. However, we consider the following classification as being of primary importance. According to the network environment in which hypermedia systems exist we distinguish between:

- Standalone hypermedia systems that provide access to a hyperweb residing on a single computer
- Large multi-user systems providing multi-point access to a hyperweb distributed over many computers connected in a network.

The focus of this work lies in the discussion of distributed hypermedia systems. A distributed environment is typically needed if either the information to be stored is too large for on machine, or if there are too many users for one machine to handle, or both. Obviously, the main prerequisite for distributed hypermedia systems is the existence of a distributed computer environment, i.e., a computer network.

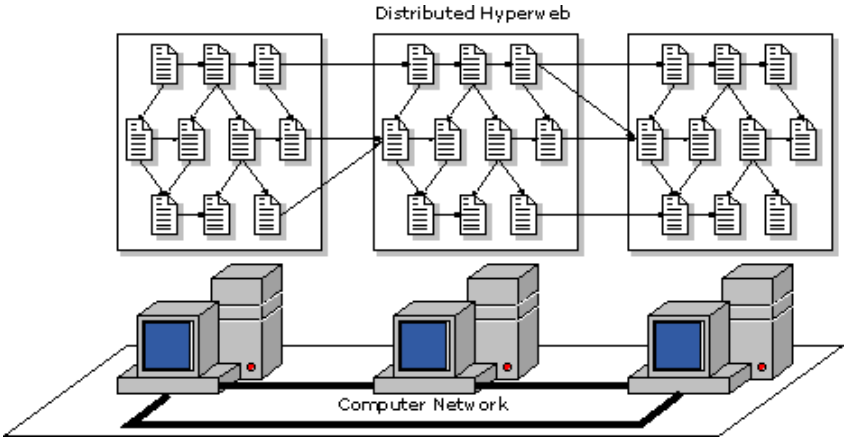


Fig. 39: Distributed Hypermedia Systems

Nowadays, the Internet is the largest worldwide distributed computer environment existing. It is in fact a network of networks that is estimated to connect several million computers and over 50 million individual users around the world - and it is still growing rapidly.
The Internet provides its users with a number of so-called Internet services such as e-mail service, file transfer service, remote login services and similar. However, the best-known and mostly used Internet service is the World Wide Web (WWW or Web). The World Wide Web is an Internet wide distributed hypermedia system. As such, it is the largest hypermedia system existing.

World Wide Web

The World Wide Web (the WWW or the Web) is the largest distributed hypermedia system nowadays. The WWW provides a remote access to the largest repository of hypermedia documents worldwide. The Web started at CERN in Geneva by Tim Berners-Lee and his team in 1991. However, the real breakthrough of the Web engaged upon the development of the first widely used Web browser, called Mosaic.
Mosaic was developed by the National Centre for Supercomputer Applications (NCSA). This browser provided a powerful graphical user interface following the simple point-and-click paradigm. Suddenly, by simply using Mosaic all documents residing on the Web were just a mouse-click away from users.

3.2 Logical Hypermedia Data Modelling

Web to become the most popular hypermedia system ever leading to a real explosion in numbers of Web sites and documents offered on the Web. Speaking more technically, the Web is based on typical client/server architecture. In the WWW, the whole system functionality is distributed over a tremendous number of computers that are interconnected by means of the Internet. The WWW utilises HTTP [HTTP, 2001] (HyperText Transfer Protocol) for client-server communication and HTML [HTML, 2001] (HyperText Mark-up Language) as a data exchange format. A particular Web server can be simply seen as a storage space for HTML and other kinds of documents where all such documents are accessible by means of so-called Uniform Resource

Locator (URL) [URL, 2001].

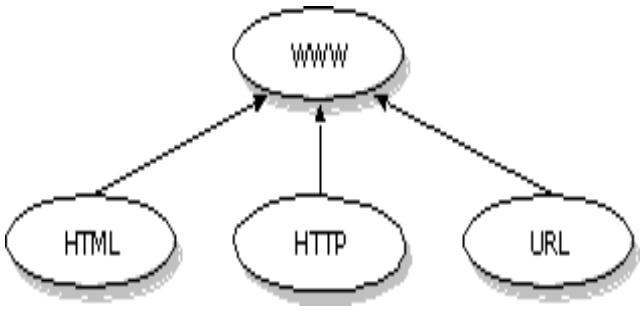


*Fig.40:*Basic WWW Concepts

WWW servers accept HTTP requests and reply, usually, with an HTML document. Note that a URL is encapsulated into such HTTP request as a particular document identity. WWW clients just access HTML documents residing on WWW servers and visualise such documents on the user screen.

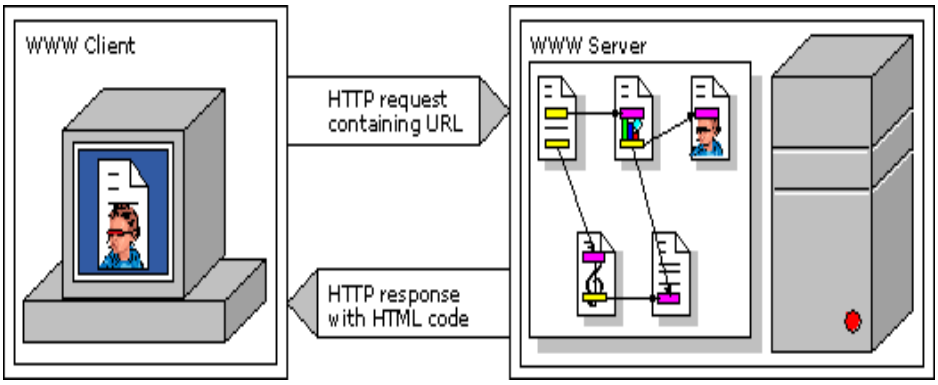


Fig. 41: WWW Architecture

Links in WWW are embedded into HTML documents as special tags containing a URL of a document, which is referred to. WWW clients are able to visualise and activate a hyperlink upon a user's action, thus requesting the linked HTML document to be visualised.
Alternatively to HTML documents WWW servers may store other kinds of documents such as: Windows Word [WinWord, 2001] documents, PowerPoint [PowerPoint, 2001] presentations, documents in Portable Document Format (PDF) [PDF, 2001], documents in eXtensible Markup Language (XML) [XML, 2001] format, etc. Recently, many of these document formats allow for embedding of hyperlinks, thus providing means for creating a hyperweb. However, in all such documents links are embedded into the documents, thus reducing the discussion of these formats on the discussion of the HTML format.

Semantic data model of the Web

The Web does not support the semantic level of the data modelling architecture at all. The focus

of the work presented in this unit lies in defining different approaches for semantic data modelling of the Web structures. These approaches are presented in chapters to follow.
However, before the presentation of those semantic data modelling approaches is given an in-depth analysis of the logical data model of the Web is required.

Current trends in logical hypermedia data modeling

In the last section a number of well-known problems of the node-link data model were presented. I listed also a number of extensions to the node-link data problem that were introduced in order to overcome these problems. In this section I provide the overview of more powerful logical data modelling paradigms and present the current trends in logical hypermedia data modelling.
As shown, the node-link data model has well-known disadvantages. However, a particular WWW server is not obliged to support the node- link paradigm internally. It can, for example, utilise the relational data model or some other data model to store data. In this case, such "advanced" WWW servers just map dynamically any incoming HTTP request into internal operations and, of course, present resultant data as HTML documents. The mapping mechanism is implemented in a so- called rendering engine running as a front-end application on a remote server [Fernandez *et al*., 1997; Andrews *et al*., 1995; Duval *et al.,* 1995].
This approach has been successfully implemented in a number of commercial products (most notably, Home [Duval *et al.,* 1995] and HyperWave).
If we compare all existing proposals for new hypermedia data modelling paradigms with the previously discussed node-link model, we may see the following main trends:

- Extending the basic hypermedia thesaurus containing notions of nodes, links and anchors with new data structures - collections, structured collections, composite components, compounds, containers or whatever. Such new data structures are addressable entities containing other data structures, nodes and links as elements [Maurer and Scherbakov, 1996; Maurer *et al.*, 1994a, Hardmann *et al.*, 1994, Streitz *et al*., 1992, Garzotto *et al.*, 1991]. I will call such data structures composites in the further discussion.
- Providing some meta-structuring mechanism (often, referred to as templates) to predefine the structure of a number of multimedia documents, thus separating the structure from the content data [Zellweger,1989].
- Providing multimedia documents with some attributes in addition to the content data [Maurer *et al*., 1998; Lennon, 1997; Andrews *et al.*,1995]).
- Upgrading links to independent, addressable objects separated from document content [Maurer *et al.,* 1998; Lennon, 1997; Maurer,1996].-Extending the notion of an anchor by associating procedures that locate documents at run-time and even dynamically generate destination documents at run-time [Maurer,1996].

In the next section I present the HM-Data model, a logical data model, far more powerful than the basic node-link data model that follows some of the mentioned data modeling trends.

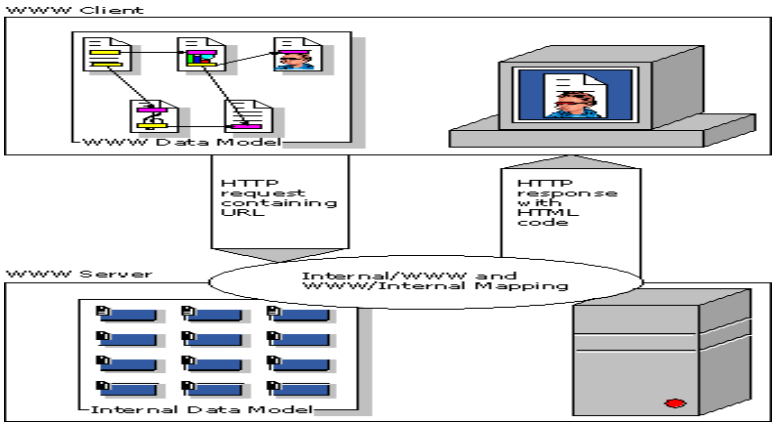


Fig. 42: Mapping Internal Data Model onto WWW Data Model

3.3 HM-Data Model

The HM-Data Model is a logical hypermedia data model. Hence, it is less concerned with the internals of a document than it is with data structures and operations that can be applied to create, modify and explore (i.e. render) such data structures. Therefore, in the following I will consider multimedia document as atomic, i.e. as basic indivisible chunks of multimedia data that have to be organised. I will examine the use of a novel method of hyper linking for structuring such information chunks into hypermedia databases (hyperweb).

3.4 HC-Data Model

The HC-Data model is a semantic hypermedia data model. As such, it is less concerned with data structures and operations that may be applied to those data structures than it is with semantic entities and relationships that exist between these entities to which data structures may be assigned. Although the HC-Data model commits to an existence of a basic data structure, namely the hypermedia composite it still provides the possibility to model hypermedia database in the terms of types of composites (entities), members (entities) and roles (relationships between entities) of members in hypermedia composites. Even, the relationship between a member and its enclosing composite ("is-part-of" or "is-member-of") may be seen as a relationship between the two entities. However, the HC-Data model does not allow defining other entity.

An application of the HC-Data model

The HC-Data model has been successfully implemented in a novel Web- based-training (WBT) [Helic *et al.*, 2000; Dietinger and Maurer, 1997; Dietinger and Maurer; 1998] system called WBT-Master [Helic *et al.*, 2001; Helic *et al.*, 2001a; Helic *et al.*, 2001b]. WBT-Master is an innovative WBT tool that supports the construction and delivery of Internet based courseware and provides all other important WBT information services on the base of the HC-Data model. In other words, WBT-Master is an Internet system that provides a set of modules and tools that use

a unified internal data structures and well-defined set of operations applicable to such data structures.

The courseware repository on WBT-Master is structured in accordance with the HC-Data model which provides for a smooth navigation through the course eliminating problems such as "getting lost in hyperspace", dangling links and similar. The model facilitates a context-dependent search and course maps. Tutors and learners may contribute to the courseware repository “on-the-fly” using such embedded mechanisms as annotations, links to external resources and multimedia attachments. All such additional elements may be defined as public, private or visible just to a group of people, and hence provide rather powerful customization facilities.

WBT-Master supports also more traditional communicational strategies such as discussion forums, brain storming sessions, chats, and exchange with private messages (ICQ). Communication may occur between learners, tutors and groups of users. Since all the communicational tools are based on the same background – the HC-Data model, any contribution may be seen as an information object, which may be stored into a courseware repository and further reused.

Illustrative example

Consider, for instance, a hypermedia database containing a large amount of computer-based educational material (courseware), where users can browse particular courses represented in hypermedia form. To be more specific, let us assume that we have a number of computers based courses prepared in the form of HC-Units: say "Course#1" and "Course#2" that all instances of the same HC-Type, say “Course”.

Each course has a title document and a number of members corresponding to chapters of the course. If a course, let us say Course#1, refers to another course, say, Course#2, the course referred to has to be inserted into the same HC-Unit as a chapter member. A chapter consists of primitive documents and refers to other courses or chapters, i.e., a chapter is an HC-Unit of another HC-Type, say “Chapter”.

It should be noted that if particular chapters and/or documents are relevant to the contents of a number of courses, the corresponding HC- Units might be re-used without any extra overhead. HC-Unit "Author- X" has a short description about the author as its title page and the author's courses (HC-Unit "Course#1", HC-Unit "Course#2", etc.) as members, i.e., the HC-Unit “Author-X” is an instance of the “Author” HC-Type.

The above discussion has indicated how primitive chunks (i.e. documents) may be gathered into more complex structures (in this case, "chapters"). The property of the HC-Data Model that an HC-Unit may belong to many other HC-Units, even recursively, provides all the necessary power to deal with more complex situations. Note the recursive membership of documents "Course#1" and "Author-X" in this example. Such recursive membership elegantly handles the common situation where a user needs to be able to access information about "Course#1" while browsing information about "Author-X" and vice versa. Moreover, if a certain chapter ("Chapter-B") refers to another course ("Course#3"), then the S-collection "Chapter-B" can be extended

with the relevant member.
To conclude our example, the HC-Units representing courses might be inserted into HC-Units dealing with particular topics (say, "Topic#1", "Topic#2" etc., which are instances of “Topic” HC-Type). Finally, one could combine the entire topic HC-Units into an HC-Unit "Library of Courseware" of the HC-Type “Library”.
To show the basic properties of this model, let us simulate the steps of a typical user session. Suppose the user accesses the HC-Unit "Library of Courseware" in some way. Suppose that the user zooms in the HC-Unit "Library of Courseware" and activates the link to the HC-Unit "Topic#1" within the "Library of Courseware". The HC-Unit "Topic#1" becomes the current member, and the chunk of multimedia information defined as its first screen template is displayed. Note that the navigational paradigm associated with the current document "Library of Courseware" is still active. The user has the possibility to access another topic by clicking on it. After selecting a particular topic, the user can zoom into this HC-Unit. Once an HC-Unit is opened the user obtains (i.e., can follow) links encapsulated within it (perhaps a menu of courses).

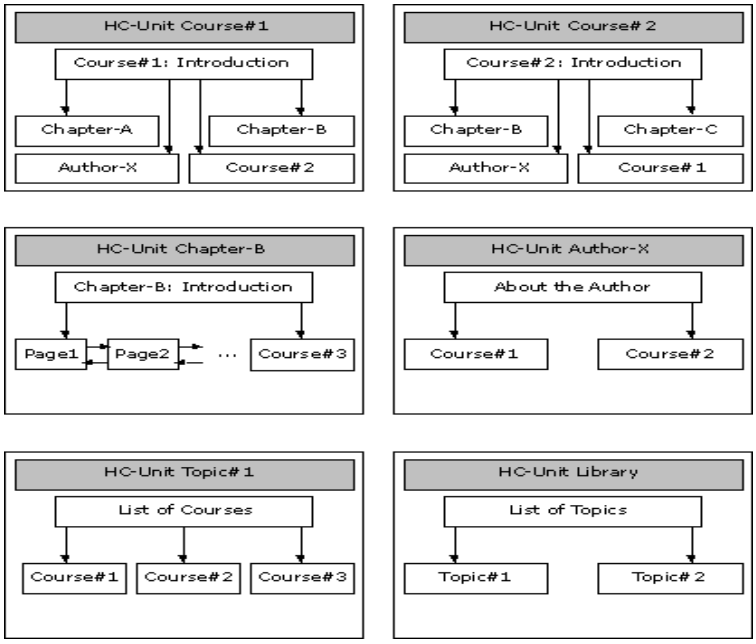


Fig. 43: Sample Hypermedia Database

Each choice from the menu results in the presentation of the first screen template of the corresponding HC-Unit. If the user has selected and entered the HC-Unit "Course#1" and then selected the HC-Unit "Chapter-B", the starting screen template, say including the abstract of the chapter is visualised ("Chapter-B" is the current member) and links to other members (i.e., to other chapters) become available. The user can either read the current chapter (i.e., open the HC-Unit) or browse other chapters. Clicking on button “Zoom Out” returns the user to HC-Unit "Topic#1". Now clicking on another member of "Topic#1" (say, on "Course#2") visualises that

member's starting screen template, clicking on button "Zoom Out" leaves "Topic#1", and returns the user to the HC- Unit "Library of Courseware", and so forth.

3.5 Approach Discussion of the Semantic Hypermedia Composites

The HC-Data model is a generalisation (an abstraction) of the HM-Data model in the sense that it extends the concept of hypermedia composites to the concept of semantic hypermedia composites providing a higher level of abstraction for different data structures involved in a particular application. Such generalisation of the HM-Data model would not be possible without an extensive use of the Data-definition-language summarised through the concept of HC-Type, i.e., a definition of a class of hypermedia composites that all share one and the same properties and the behavior.

Hence, the HC-Data Model does not itself define a number of different classes of hypermedia composites, but rather it gives administrators facilities to define new composite types useful for a particular application.

All that is said for the HM-Data model and its advantages over the node- link data model is still valid in the case of the HC-Data model (i.e., in the case of a generalized HM-Data model). Thus, we discuss here rather the improvements of the HC-Data model over the HM-Data model itself.

The advantages of the HC-Data Model over its logical data-modelling pendant can be stated as follows:

- Possibility to apply purpose-oriented or application-specific data structures. For example, WBT-Master, being a typical WBT application, defines a wide range of data structures typical for a WBT system. Those data structures include learning units, learning course, discussion forum and similar. Such data structures are easier to apply by all users of such a system. For instance, for authors in a typical WBT system is far more intuitive to manipulate learning units or learning courses, than it would be to manipulate S-Collections.

  On the other hand, learners browse the content of such hypermedia database; the possibility to browse those data structures improves the possibility of learners to comprehend the educational material in the right way to say at least. Thus, users work with data structures that are much closer to real-life objects rather than with such generic data structures as folders, envelopes, etc. This facilitates improved overall understanding of the purpose of a particular hypermedia system.
- Template- based authoring decreases tremendously authoring effort. The concept of the definition of a type of hypermedia composite (HC-Type) provides the base for a template-based authoring. Again, WBT-Master implements such a facility on the full extent. It allows authors a rapid production of qualitative educational material.
- Inheritance mechanism allows for the creation of a wide range of instances of one end of

the same type. All of these instances share common properties and behaviour.

- Through the concept of properties that are defined in an HC- Type, documents are provided with useful meta-data. Those meta-data may be used to provide users with useful navigational tools such as meta-data search engines.

Such advantages provide means for supporting knowledge structuring process in hypermedia systems. Thus, a resultant hyperweb is composed of a number of well-structured knowledge chunks of hypermedia information

On the other hand, we might also see a number of disadvantages. Let us consider the following example. One of the basic concepts of the HC- Data model can be defined as imposing different types (HC-Types) of data structures on top of existing collections of HTML documents and/or other data structures. As it was the case with the HM-Data model, this allows a satisfactory level of reuse of documents and composites in different contexts.

However, all those previously discussed data structures such as hypermedia composites, different HC-Types, HC-Units were “navigational oriented” so to speak. They mainly can be perceived as different navigational paradigms, reflecting mainly different ways of accessing and working through a particular hyperweb by users of hypermedia systems. Primitively speaking, a "navigational oriented" data structure consisting of documents "B" and "C" mainly prescribes reading "B" before reading "C" and has nothing to do with a possible situation that “C” may be a documentation on a software module implemented by the programmer "A" for a project "B". Often, users need a general overview and access to all documents provided by a particular hypermedia system.

Let us just discuss the following situation [Helic et al., 2001a]. Suppose a software organisation maintains a big repository of technical documents in the form of a WWW application. Obviously, elements of the repository are valuable resources and may be reused as components of different composites. At the same time, localisation of a particular document may constitute a rather difficult problem which can be only solved by structuring the repository on meta-level invariantly to any navigational paradigm. Using the same primitive language as before, we can say that the knowledge: "C" is a technical description of the software module implemented by the programmer "A" for the project "B" should be kept independently of reusing "C", "B" and "A" in different contexts.

As we see, a possibility to define richer network of entities and relationships between such entities, which is absent in the HC-Data model may be seen as being of primary importance. As already mentioned, HC-Units and their members may be seen as entities, which are related by means of the “is-part-of” relationship. Often, in order to model a particular hypermedia database on the semantic level we need to say more about the contents of this database (e.g., "C" is a technical description of the software module implemented by the programmer "A" for the project

"B").
Actually, the described situation reflects the process of knowledge profiling. Thus, the HC-Data model is not expressive enough to provide means for creating a hyperweb with profiled knowledge attached to it.

Knowledge representation in hypermedia systems

First of all, the possibility to define new types of composites, with a possible different member roles and navigation and visualisation paradigms may be seen as a great advantage of semantic hypermedia composites over their logical pendant. Such possibility provides for a resulting hypermedia database structured in accordance to the application-specific or purpose-oriented data structures, i.e., in accordance to the previously defined composite types. A hypermedia database structured in such manner is much more easily created, maintained, or accessed/browsed by a wide range of different users.

Unfortunately, a hypermedia database based on the concept of semantic hypermedia composites has also some limitations. A most important limitation of such an approach is the lack of facilities to express the semantic knowledge implicitly contained in such a database. In other words semantic hypermedia composites allow creating data structures that are highly “navigation oriented”. That means that such an approach is navigation- centred, so to speak.

The navigational structure is put into the centre, either as the matter of the authoring process (i.e., the navigational structure should be easily created and maintained) or as the matter of the information retrieval process (i.e., the navigational structure should be intuitive and easily comprehended). However, semantic hypermedia composites do not provide a possibility to structure the hypermedia database at a global semantic level. It is not possible to provide users with a general overview of the hypermedia database, i.e., with a profile of knowledge contained in such database. For instance, semantic hypermedia composites does not allow to express the fact that the document “C” is a technical description of the software module implemented by the programmer "A" for the project "B", but merely they prescribe that the document “B” should be read before the document “C”.

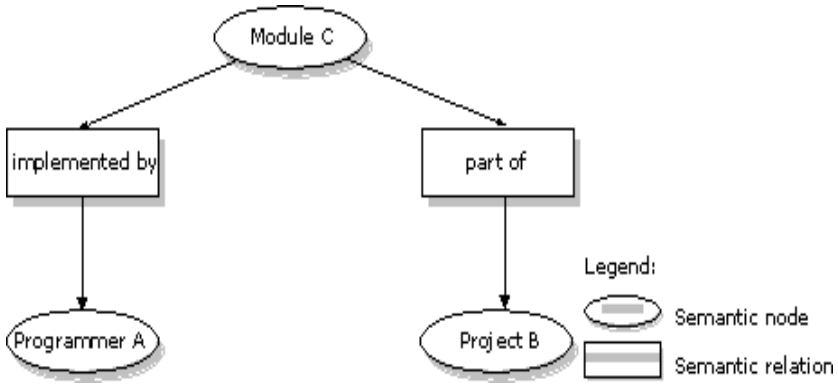


Fig.44: Semantic Network of Resources in a Hypermedia Database

Properties of semantic networks

A hypermedia database structured accordingly to a knowledge domain schema is a repository of well-structured reusable hypermedia modules enhanced with a semantic network of those modules. The semantic network represents knowledge contained in those hypermedia modules in the form of different concepts, to which hypermedia modules may be assigned, and relationships between these concepts.

Generally, semantic networks may be used as a component of two different information-processing tasks:

- **Information Retrieval:**
  - **User Interface**: Semantic network may be seen as a simple and intuitive visual form of knowledge representation and, thus, as a metaphor for man-machine communication [Sowa, 1984; Sowa, 1991].
  - **Querying**: Semantic network may be seen as a database schema that allows for querying the concepts and their relationships [Sowa, 1984; Sowa,1991].

- **Inference Engine:** Semantic network may be seen as a special formalism for defining rules of inference [Sowa, 1984; Sowa, 1991].

A Knowledge Domain applies a semantic network to create a convenient and intuitive graphical user interface for browsing the hypermedia database, as well as a powerful search mechanism to search for hypermedia modules represented by concepts from a particular semantic network.

However, Knowledge Domains does not make use of the other very important property of semantic networks, i.e., the possibility to infer a new knowledge from the semantic network by an application of so- called inference rules. The concept of knowledge cards tries to make use of the inference power of semantic networks. Furthermore, this concept tries to combine the information retrieval facilities of semantic networks with an inference engine based on a number of very simple inference rules.

Such a global semantic structuring may be accomplished only by means of more powerful knowledge representation mechanisms, which include a conceptual structuring of the subject matter. Such mechanisms usually support the definition of so-called domain ontology, which defines concepts, attributes and relationships between such concepts in accordance with a particular problem domain.

The hypermedia database is structured corresponding to a number of such definitions, where particular document and/or composites from the hypermedia database are assigned to an arbitrary number of concepts and relationships. The outcome of such structuring technique might be for instance represented by means of network knowledge representations in the form of a semantic

network of hypermedia nodes containing a semantic knowledge about hypermedia nodes and relationships between such nodes.

Such graphical representations [Gains and Shaw, 1995] facilitate greatly information retrieval techniques usually applied in hypermedia systems: browsing [Gains and Shaw, 1995; Brusilowski and Schwarz, 1997; Barwise and Etchemenedy, 1990; Chang *et al*., 1986]. Of course, querying of such data structures [Arents and Bogaerts, 1996; Comai *et al.*, 1998] might be also easily facilitated.

3.6 Introducing Semantic Data Structures to the WWW

The semantic hypermedia data modelling concepts that were introduced in the last chapter were successfully implemented in the novel Web- based-training system called WBT-Master [Helic *et al*., 2001; Helic *et al.*, 2001a; Helic *et al.*, 2001b]. This chapter gives an overview of implementation issues of those data modeling approaches.

3.7 WBT-Master

The WBT-Master is a novel Web-based-training system implemented on the following concept: a modern WBT system should provide a set of tools which support different knowledge transfer processes, thus allowing for a smooth transfer of knowledge from people who possess such knowledge (e.g. experts, teachers) to people who want to acquire this knowledge (e.g. learners, students) [Helic *et al.*, 2001b]. Thus, WBT-Master provides tools that support the following knowledge transfer processes in a Web based environment [Helic *et al.*,2001b]:

- Web based knowledge structuring
- Web based knowledge mining
- Web based knowledge profiling
- Web based tutoring
- Web based mentoring
- Web based learning.

Thus, WBT-Master was an implementation of the previously discussed knowledge transfer processes in a Web environment. Here we provide the overview of such implementation.

WBT-Master architecture

Being a fully WWW compatible, WBT-Master considerably extends the standard WWW client-server architecture. WBT-Master servers store complex, composite data objects additionally to primitive HTML documents. The data objects are typed, i.e. a data objects always belongs to a particular data class, which defines all user's actions applicable to such data objects. Documents, portals, learning units, learning courses, learning goals, etc. are data objects residing on WBT-Master server.

The data objects are persistent, i.e. a WBT-Master server can apply so- called actions, which

alter a current state of a data object, and results of such actions are available to other users. For example, a new member may be inserted into a learning unit, a new contribution added to forum, etc.

The data objects are reactive, i.e. an object replies to an action with a collection of data which depends on the current state of the data object, and user's context. For example, "get content" action addressed to a discussion forum, returns a structured collection of contributions made to this forum.

Actions are sent from an Internet client to the WBT-Master using ordinary HTTP protocol. Note that the HTTP requests contain not only the URL of a particular data object but also an action, which is to be applied to this data object. The WBT-Master server carries out the request, which results in:

- Possible modification of the target object current state
- Generating a response to the action.

The server response is visualised on the WBT-Master client side as resultant information and all further actions, which can be applied to such data object (as opposed to a visualisation of passive HTML documents).

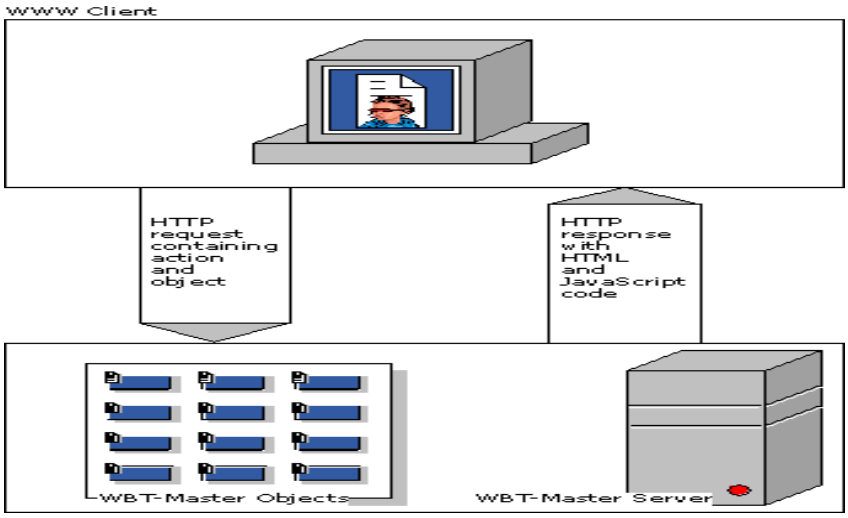


Fig. 47: World Wide Web Client

All the previously discussed data structures, such as Knowledge Domain, Knowledge Cards or HC-Types and HC-Units are different kinds of WBT-Master objects. Thus, WBT-Master provides an implementation of the previously discussed semantic data modelling approaches.

WBT-Master technical solutions

Recollect the basic WBT-Master architectural principles [WBT-Master, 2001]:

- Data Objects are persistent, reside on a server and are eligible for actions that may alter

the data objects.

- A particular data object belongs to one of predefined data types that define properties of the data object and valid actions.
- A client initiates an HTTP request containing reference to a target data object, an action that need to be applied to it and a number of parameters relevant to this action.
- A server applies the action to a prescribed data object. The action may affect the object's current state. The response to the action is sent back to the client where it is visualized on the user's screen.

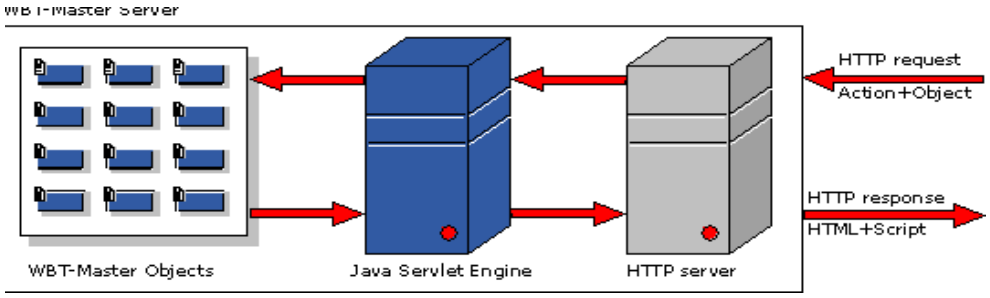


Fig. 48: WBT-Master Server Architecture

Obviously, standard WWW servers and clients do not support the above-mentioned functionality. Hence, the functionality of both components (i.e. server and client) must be extended by means of one of modern WWW programming technologies. On the server side WBT- Master extends a standard WWW server functionality by applying so- called Java Servlet (a Java-enabled Web server). Since Java Servlets are small Java programs running in a context of a Java Servlet Engine, provide an ideal mechanism for implementing WBT-Master actions.

Current implementation of the WBT-Master is based on Apache Web Server with the Apache Servlet module enabled and JServ Servlet Engine. However, any server that supports server site Java Applets (i.e. Servlets) can be used as well. An Apache Web Server acts as a repository of data objects. It stores data objects in its file system. A client request including an encapsulated action is interpreted by a Servlet which actually generates a response depending on the current object state.

The current servlets implementation assumes that data objects are instances of an abstract Java class. This basic abstract class is defined through its method interface that represents actions that can be applied to data objects. Subclasses of the basic abstract data object class support particular logic of an action applied to a predefined type of a data object.

This level of abstraction enables dynamical mapping of logical data structures onto a variety of possible physical storage spaces. For example, an ordinary Apache file system may be replaced with a database by implementing a new subclass of the basic abstract class.

Moreover, since the servlets utilise the Java Factory concept it is possible to change the physical data objects format even at the runtime. Conceptually, an end-user always communicates with a particular instance of data object. Thus, for example, the user may work on a particular learning course, learning goal, forum, etc.
On the client side, JavaScript functions and signed Java applets are used to:

- Visualise all actions applicable to a current data object
- Convert a particularly selected action into an HTTP request
- Visualise the action's results (i.e. server response) on the user's screen.

4.0 Self-Assessment Exercise(s)

1. What do you understand by the term Hypermedia?
2. Mention and explain the classification of hypermedia.

5.0 Conclusion

With exploration of the application of semantic data modelling to computer world, it is now clear that, interconnected information is fundamental to hypermedia. So, semantic data models are particularly effective in capturing the complex inter-object relationships that characterize hypermedia; its techniques also also allow structure, information, and behaviour to be abstracted from hypermedia.

6.0 Summary

In this unit, you learnt that:

- World Wide Web started as a small Internet based hypertext project at CERN (European Organisation for Nuclear Research) in late1990
- a typical hypertext consists of a number of chunks of textual information, usually in hyper nodes or simply nodes
- hypermedia systems can be classified as frame-based, window- based, standalone and large multi-user system
- the HC Data Model is a generalisation (abstraction) of the HM Data Model, in the sense that, it extends the concept of hypermedia composites to the concept of semantic hypermedia.

7.0 Further Readings

Al-Khatib, W. *et al*. (1999). Semantic Modeling and Knowledge Representation in Multimedia Databases. IEEE *Transactions on Knowledge and Data Engineering,* 11(1), 64-80,1999.

Andrews, K. (1996). "Browsing, Building and Beholding Cyberspace: New Approaches to the Navigation, Construction and Visualisation of Hypermedia on the Internet" PhD. Thesis, Graz University of Technology, Austria,1996.

Andrews, K. *et al*. (1995). Andrews, K.; Nedoumov, A. and Scherbakov, N., Embedding Courseware Into Internet: Problems and Solutions, Proceedings of ED- MEDIA'95, Graz, Austria, 69- 74,1995.

Arents, H.C & Bogaerts, W.F.L. (1996). Concept-Based Indexing and Retrieval of Hypermedia Information, Encyclopaedia of Library and Information Sciences (supplement volumes). Vol. 58, 1-29, Academic Press, New York: [ASP, 2001] Active Server Pages (General).

Barwise, J & Etchemenedy, J. (1990). Valid Inference and Visual Representation, Zimmerman & Cunningham (eds). *Visualisation in Mathematics*. American Mathematics Association.1990.

Berners-Lee, T.; *et al*. (1992). World-Wide Web: *The Information Universe, Electronic Networking, Research, Applications and Policy,* 1(2), 74-82, 1992.

http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000440

Hills, T. (2019). *NoSQL and SQL data modeling: Bringing together data, semantics, and software*. New York: Technics publications.

Hussein, T., Paulheim, H., Lukosch, S., Ziegler, J., & Calvary, G. (2016). *Semantic models for adaptive interactive systems*. London: Springer.

Qin, Z., & Tang, Y. (2014). *Uncertainty modeling for data mining a label semantics approach*. Hangzhou: Zhejiang Univ. Press.

Tutcher, J. (2015). *Development of semantic data models to support data interoperability in the rail industry*. Birmingham: University of Birmingham.

Uzun, A. (2019). *Semantic Modeling and Enrichment of Mobile and WiFi Network Data*. Cham: Springer International Publishing.

Unit 2: APPLICATION IN BUSINESS

Contents

1.0 Introduction

In today's business ecosystems, information has become a competitive and strategic asset. Being able to exchange data and to interpret this data in the right context and within a reasonable time is a top priority for many organisations. And adding semantics and business context to your integration architecture will reduce complexity, increase agility, and improve governance. This integration model is considered the best- practice when integrating applications on an Enterprise Service Bus.

The challenges of implementing this architecture successfully are threefold:

- lack of semantic alignment
- lack of flexibility, and
- lack of governance.

These challenges are discussed in details, and answers are provided on how to tackle these issues by:

- adding business context to your disparate data sources and let it drive the integration process
- involving technical and business stakeholders by de-coupling structure from meaning in terms of business vocabularies, facts and rules
- leveraging these business semantics operationally by deploying them as data services on your Enterprise Service Bus.

2.0 Intended Learning Outcomes (ILOs)

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

- state the challenges of sharing data models
- describe the business semantic for application integration
- describe the business semantics management product suite.

3.0 Main Content

3.1 Challenges of a Shared Data Model

With these ambitious goals, in a complex environment as application integration, it is quite obvious that achieving these benefits comes with quite some challenges. Three gaps have been identified in current solutions that will need to be overcome: Lack of semantic alignment, lack of flexibility and lack of governance.

Lack of semantic alignment

From a conceptual and business perspective, the structure and the meaning (read: semantics) of a data model are completely unrelated. Data models are usually created in XML or UML. These Models have a well-defined structure and relations, but do not sufficiently define the meaning of the information concepts. Furthermore, they are attributed the status shared because, they are modelled in a standard format, and not necessarily because they are the product of an agreement in a community of practice.

If we take XML as an example, the following quote describes the limitations of XML as a means to capture the semantics of data:

- "While the definition of an XML protocol element using validity formalism is useful, it is not sufficient. XML by itself does not supply semantics.

Any document defining a protocol element with XML MUST also has sufficient prose in the document describing the semantics of whatever XML the document has elected to define." RFC 3470, "Guidelines for the Use of XML within IETF Protocols" January 2003 Even if we assume a model is shared, a merely technical data model does not sufficiently grasp the contextual meaning of business assets in the organisation. These results in semantic gaps over three different dimensions: from systems to systems, from people to people, and from people to systems. A shared data model in XML or UML covers none of these gaps:

- **Systems to Systems:** While the disparate systems can be connected through a shared, but technical, data model, that model does not supply any semantics. The semantic gap between these systems can therefore not be identified nor solved.

- **People to Systems:** Given the lack of semantics in such a shared data model, it is still subject to interpretation from the different stakeholders. A shared data model does not reduce the risk of interpretation errors leading to erroneous transformation mappings, model use, etc.
- **People to People:** By not making the interpretation of the different data formats explicit, the traditional misalignment between different stakeholders (business and IT) is not solved. This leads to increased project costs through miscommunication and repeated work.

"The market lacks tools capable of rationalising business process models, logical information models and repository management tools for automated semantic resolution in SOA."

Lack of flexibility

The flexibility of your shared data model is crucial to cope with what we call the coverage of your disparate systems. The goal should be to have a shared model that covers 100% of the systems to be integrated. The following illustration provides an example:

Lack of governance

When your shared data model is polluted by application-specific structure and does not provide the necessary domain semantics, it remains on a technical level. This is sufficient for basic governance like impact analysis. But it goes much further than that. The goal should be to push the shared data model to the same level as business process management tries to do with processes. To keep this shared data model alive and valuable for the organisation, it is imperative to involve all its stakeholders: the domain experts (usually the business (analysts)), the functional analysts, and the technical experts. If you involve business stakeholders, they will want to be able to have a say in the process.

Assigning concept stewards is a great way to involve these users and keep the shared data model alive and up-to-date with the ever-changing business requirements. Concept stewardship, however, doesn't work well on models defined on a technical level. Another important governance aspect is compliancy with industry standards. When directly using one standard, you are technically bound to this inflexible, static standard. This makes compliance with other standards, or adding new concepts and relations to your shared data model, technically and operationally complex. Developing a semantic model from a relevant business context perspective instead of an application or standard perspective will add a valuable de-coupling layer that will provide compliancy with industry standards and remain flexible and efficient in its use.

3.2 Business Semantics for Application Integration

Not acting on the three challenges identified before (lack of semantic alignment, lack of

flexibility, lack of governance), will not only increase the risk of project failure it also limits the potential up-side of your investment. Adopting a shared information model is a great opportunity to better align business & IT, increase information governance through transparency and traceability, and create a true enterprise asset, without additional costs.
The vision is to help companies achieve semantic intelligence.
Similar to Operational Intelligence or Business Process Intelligence, which aims to identify, detect and then optimise business processes, semantic intelligence targets information instead of processes. It aims to enable better understanding and provide insight in data for all stakeholders. Semantic Intelligence supports better information sharing, reuse, governance and better decision-making.

What we will describe here is how taking steps towards that goal will solve the three major challenges of adopting a shared information model for application integration. The core of this approach is to separate the meaning from the structure of the shared information model. Doing so enables all stakeholders to understand the shared model, increases flexibility through additional de-coupling, and promotes governance and manageability by enabling each stakeholder to govern its own interest.

With respect to metadata, four different levels can be identified, namely the conceptual-, relational-, technical-, and operational level.
The conceptual layer describes the business concepts in the organisation, understandable for any stakeholder familiar with the domain. In practice, these are often stored in dictionaries, vocabularies, and taxonomies.

A second layer is the relational layer where facts, relations and patterns of these business concepts are managed. These help the business to put business concepts into context, see how these concepts are used, and how they relate to other concepts. A third level is the technical level. This level describes the concepts and relations from the two levels above, but in an application-specific way. Typical standards used on this level are UML and XSD (XML Schema). These data models consist of sufficient application semantics to be directly linked to applications or to be used for model-driven development. The operational level is where the actual data sources are located.

Currently, metadata on all four of these levels is mostly managed in an ad-hoc manner. Within a certain layer, there is often no linking, harmonisation, reuse or reconciliation between disparate metadata models. The only place where this does occur is on the technical level, using metadata repositories (usually several disconnected ones). These repositories however, are very tool-dependent and fail to reach sufficient coverage. They don’t harmonise nor reconcile metadata on the other levels. Interestingly, the more metadata moves from a technical to a business context, the more it becomes ad-hoc. Business people usually use typical office tools like word

processors, document managing systems or spreadsheets to manage business-critical metadata.

3.2.1 Semantic Alignment

It is imperative to push the shared information model up to the relational and conceptual levels. Doing so will enable you to de-couple the application-specific structure of your data model from the meaning of the business concepts. On the conceptual level, it enables the business to collaboratively capture and define the business concepts relevant for the problem domain through simple definitions, examples, synonyms, etc.

On the relational level, it enables the business and technical stakeholders to think in a fact-based manner how these business concepts relate to each other. These business facts can be grouped together into a semantic pattern which captures the semantics of a certain problem scope on which all stakeholders can easily agree. This semantic pattern is, in this case, the conceptual representation of your shared information model. Technical models such as UML, XSD or database schema's can be automatically generated from a semantic pattern through an automatically generated commitment.

The approach illustrated above, enables you to cover all four metadata levels. This in turn will help you to involve all stakeholders in the process. Because all stakeholders are involved, the result will be of higher quality and will remain up-to-date from a business as well as a technical perspective.

3.2.2 Flexibility

The core flexibility problem in adopting a shared information model lies in the big impact of necessary changes to the shared technical model on all the other existing applications. On the one hand, it should be possible to easily change the shared model as needed to get the new integration requirements finished in time. On the other hand, the shared model should be well governed or it will explode in size and complexity, and will quickly become so unpractical it becomes just another data format to integrate.

The flexibility to change the shared information model as needed, and the ability to govern these changes effectively.

Key in this approach the point where the reconciliation process and the application process come together: The unify activity. Collibra's Business Semantics Management product suite features a feedback loop through Collibra's Business Semantics Enabler product.

3.2.3 Governance

When adopting a shared information model for application integration, being able to effectively govern that model is crucial to the project's success. Collibra introduces a closed-loop feedback technique to make sure all different stakeholders can act effectively on their responsibilities:

- The technical stakeholders can make the necessary changes when the shared model is insufficient to solve the integration problem.

- The business stakeholders and domain experts can make sure the shared model correctly describes the business context.
- The business stakeholders get involved by appointing concepts stewards to govern the concepts described on a business level. The technical stakeholders create and govern the commitments between the business context (the semantic pattern) and the underlying applications. These commitments are pushed up to the business level through a closed-loop feedback mechanism. Collibra's patented technology enables all stakeholders to collaboratively govern the changes in these semantic patterns. This provides the necessary versioning, transparency and traceabilityfunctionality.

3.3 Example from the Supply Chain Industry

As a practical example, we present an implementation of Business Semantics for application integration at SCA Packaging. SCA packaging is a leading European provider of customer-specific packaging solutions with 220 production units in 30 countries and 15500 employees. SCAi is SCA's integration platform as well as a competence centre, across all SCA packaging divisions and regions, dedicated to maintaining a prompt and cost-effective response for all integration needs. Some figures on the current SCAi setup: 400 different message formats, 600 mappings since start up, 3,500 routes, approximately 400 nodes, a core team of 6 FTEs.

SCA packaging is a member of the papiNet standard group, a global community involved in supply chain processes for the forest and paper industries. The papiNet group has developed a standard XML schema that should represent the business concepts in this industry. However, as we have described above, this standard quickly becomes just one of the systems to be integrated, next to the hundreds of EDI, XML or database systems.

Semantic data integration solution enables SCA to:

Reduce the complexity and number of mappings by committing to a shared business context / semantic model. Increase the flexibility of the solution by adding a de-coupling layer between different standards and message formats. Increase efficiency by allowing people to effectively communicate and agree on the meaning of the business context and reuse these concepts in different integration scenarios. Increase agility and governance by involving domain experts (usually business analysts or enterprise architects) to define and govern the business context which drives the integration process.

The different applications are no longer integrated by ad-hoc, point-to- point mappings but through a shared semantic model which describes the business context. It also shows how this additional de-coupling supports different and changing standards or pre-existing internal data models in a technical format such as XSD or UML.

Furthermore, the three transparent rectangles show how the process of agreeing on the meaning of the business assets in their business context can be effectively achieved by domain experts

without the additional application-specific format complexity. The technical experts doing the actual integration have a hard time agreeing on a shared meaning because they think in terms of their application-specific data formats. Using our approach, they can simply connect, or what we call commit their application-specific formats onto the shared business context.

Collibra Software

Collibra is an enterprise software company integrating with and adding Business Semantics to all major SOA, EAI & Data integration vendors. Business Semantics Management defines the specific meaning and context of key business assets (e.g. 'customer', 'product', 'interest rate',...) for organisations, converting them into clearly defined business facts and rules compiled in a 'Business Glossary'.

The solution translates these key business assets into executable models that drive EAI/SOA, Business Intelligence, Semantic Data Integration and Metadata based on relevant business context instead of pure system to system integrations.

3.4 Business Semantics Management Product Suite

Business Semantics Glossary

The Business Semantics Glossary is a web-based product aimed at both business as well as technical users. It lets people collaboratively define and govern the meaning of the business assets in their business context. It is the first enterprise-ready tool that implements the OMG's SBVR industry standard (Semantics of Business Vocabulary and Business Rules) to define business vocabularies, facts and rules.

Business Semantics Studio

The Business Semantics Studio is an Eclipse-based tool suite that enables IT professionals to tie meaningful business context to the technical models and (legacy) data sources. It provides modelling functionality to extract semantic patterns from a business context. It also provides mapping functionality to commit existing data sources onto these semantic patterns.

Business Semantics Enabler

The Enabler is a run-time server that leverages your IT infrastructure by generating data services based on shared contextual meaning. The Information Enabler enables automatic semantic integration possible: given a set of business semantics, the Information Enabler can automatically trans-form data between different data formats or web- services.

4.0 Self-Assessment Exercise(s)

1. What are the challenges of sharing data model?
2. Mention some of the Business Semantics Management product suite.
3. What do you understand by the term Flexibility in Business Semantic Management?

5.0 Conclusion

Being able to exchange data and to interpret the information in the data that has been exchanged in the right context and within a reasonable time is a top priority for many organisations. In this white paper, we have identified three critical challenges when adopting a shared data model to drive application integration:

1) **Lack of semantic alignment:** Technical data models do not provide the business context or the semantics of the data. It does not solve the semantic gaps between people, between systems, and from systems to people.
2) **Lack of flexibility:** A technical data model is application specific. This makes it extremely complex and inefficient to cover all the semantic differences of the different applications in a single data model or manage different shared data models that cover these semantic differences.
3) **Lack of governance:** A shared model should involve all its stakeholders, business and IT. The business context that drives the integration should be well governed and kept alive. In this lecture note, we have laid out several solutions to cope with these challenges:
    a. Adding business context to your disparate data sources and let it drive the integration process,
    b. Involving technical and business stakeholders by de-coupling structure from meaning in terms of business vocabularies, facts and rules.

6.0 Summary

Many organisations have chosen to adopt a shared or so-called canonical data model for their data and application integration. And that, such an integration pattern, in theory, would increase the efficiency, agility and transparency of application integration for the organisation. As a result, adding business context and semantics to the existing infrastructure in this manner, enables organisations to leverage investments by increasing agility and governance and reducing complexity.

7.0 Further Readings

ACM Trans. Database Syst. 6, 3 (Sept. 1981), 351−386. 25.

Garcia, R. & Celma, O. Semantic Integration and Retrieval of Multimedia Metadata. In Proc. of the 5th International Workshop on Knowledge Markup and Semantic Annotation (SemAnnot 2005), Galway, Ireland:

Hammer, M. & McLeod, D. Database Description with SDM: A Semantic Database Model.

Hudson, S. E. & King, R. (2005). The Cactis Project: Database Support for Software Environments. *IEEE Trans. Softw. Eng.* 14, 6 (June 1988), 709−719. November 2005.

Peckham, J. & Maryanski, F. Semantic Data Models. *ACM Comput. Surv.* 20, 3 (Sept.1988), 153−189.

Potter, W. D. &Trueblood, R. P. Traditional, Semantic, and Hyper- Semantic Approaches to Data Modelling. Computer 21, 6 (June 1988), pp. 53−63.

www.collibra.com tangible solutions for your business semantics.

Hills, T. (2019). *NoSQL and SQL data modeling: Bringing together data, semantics, and software*. New York: Technics publications.

Hussein, T., Paulheim, H., Lukosch, S., Ziegler, J., & Calvary, G. (2016). *Semantic models for adaptive interactive systems*. London: Springer.

Qin, Z., & Tang, Y. (2014). *Uncertainty modeling for data mining a label semantics approach*. Hangzhou: Zhejiang Univ. Press.

Tutcher, J. (2015). *Development of semantic data models to support data interoperability in the rail industry*. Birmingham: University of Birmingham.

Uzun, A. (2019). *Semantic Modeling and Enrichment of Mobile and WiFi Network Data*. Cham: Springer International Publishing.