FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA
NIGER STATE, NIGERIA

CENTRE FOR OPEN DISTANCE AND
e-LEARNING (CODeL)

B.TECH. COMPUTER SCIENCE
PROGRAMME

COURSE TITLE

DATABASE DESIGN MANAGEMENT

COURSE CODE

CPT325

COURSE CODE

CPT325

CREDIT UNIT

2

Course Coordinator

Bashir MOHAMMAD (Ph.D.)

Department of Computer Science

Federal University of Technology, (FUT) Minna

Minna, Niger State, Nigeria.

Course Development Team

CPT 325 DATABASE DESIGN MANAGEMENT

Subject Matter Experts	Sapun AKSANA (Mrs.) Popapenko NATALYA Computer Science Department FUT Minna, Nigeria.
Course Coordinator	Bashir Mohammad (Ph.D.) Computer Science Department FUT Minna, Nigeria.
ODL Experts	Amosa Isiaka GAMBARI (Ph.D.) Nicholas Ehikioya ESEZOBOR
Instructional System Designers	Oluwole Caleb FALODE (Ph.D.) Bushrah Temitope OJOYE (Mrs.)
Language Editors	Chinenye Priscilla UZOCHUKWU (Mrs.) Mubarak Jamiu ALABEDE
Centre Director	Abiodun Musa AIBINU (Ph.D.) Centre for Open Distance & e-Learning FUT Minna, Nigeria.

CPT 325: Study Guide

Introduction

CPT 325 Database Design and Management is a 2 Credit Unit course for students studying towards acquiring a Bachelor of Science in Computer Science and other related disciplines. The course is divided into 4 modules and 15 study units. It will first introduce an overview of database management system basic concept. Then the history of database management system, types, functions. Entity Relationship model and Relational model will also be discussed. Thereafter database query language using relational calculus, relational algebra and SQL follows. Finally, distributed database as well as other vital topics will be discussed.

The course guide therefore gives you an overview of what the course; CPT 325 is all about, the textbooks and other materials to be referenced, what you expect to know in each unit, and how to work through the course material.

What you will learn in this Course

The overall aim of this course, CPT 325 is to introduce you to basic concepts of database design and management in order to enable you to understand the basic database concepts that are used in building today's sophisticated application.

Course Aim

This course aims to introduce you to the basics, concepts and design of database management system. It is believed the knowledge will enable the reader understand the basic database concept used in developing information system and hence know how to build a reliable, dependable and efficient system.

Course Objectives

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

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

1. Explain the basic concepts of Database management System
2. Define a Database and Database Management System
3. Describe the Database Management structure
4. Mention and explain components of database management system
5. Define basic foundational terms of Database
6. State the advantages and disadvantages of the different models
7. Formulate queries using relational calculus, relational algebra and Structured Query Language(SQL)

8. Discuss the constraints and controversies associated with relational database model.
9. Explain functional dependency and normal form
10. Compare and contrast the types of RDBMS based on several criteria
11. Explain the concept of data planning and Database design
12. Describe the steps in the development of Databases
13. Explain distributed databases
14. Discuss the knowledge of transaction processing

Working through this Course

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

Course Materials

The major components of the course are:

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

Study Units

There are 13 study units and 4 modules in this course. They are:

Module 1: Overview

UNIT 1 Introduction to Database Management System

UNIT 2 Classification of Database Management System

UNIT 3 Component of Database Management System

UNIT 4 Database Management Architecture & Data Independence

Module 2: Database Model

UNIT 1 Entity Relationship (ER) Model

UNIT 2 Relational Database

UNIT 2Functional Dependency &Database Normalisation

Module 3: Database Languages

UNIT 1Database Languages

UNIT 2 Relational Algebra

UNIT 3Relational Calculus

UNIT 4Introductions to Structural Query Language (SQL)

UNIT5 Structural Query Language Commands

Module 4: Other Topics

UNIT 1 Transaction processing

UNIT 2 Overview of Distributed databases

UNIT 3 Physical database design

Recommended Texts

These texts and especially the internet resource links will be of enormous benefit to you in learning this course:

1. Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill
2. Connolly, T., Begg, C., and Strachan, A. (1998) *Database Systems, A Practical Approach to Design, Implementation, and Management*, Addison- Wesley, Harlow, England.
3. Earp, R. and Bagui, S. (2000) *"Building an Entity Relationship Diagram: A Software Engineering Approach," Database Management*, Auerbach Publications, Boca Raton, FL, 22-10-41, 1–16,
4. Elmasri, R. and Navathe, S.B. (2000) *Fundamentals of Database Systems*, 3rd ed., Addison-Wesley, Reading, MA.
5. Fred R. M., Jeffrey A.H., Mary B. P. (2005). Modern Database Management (7th ed.), Prentice Hall.
6. Ramakrishnan, R. and Gehrke, J., (2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.
7. Sanders, L., (1995) *Data Modeling,* Boyd & Fraser Publishing, Danvers, MA
8. Shouhong W., & Hai W., Database Design and Implementation: Florida: Universal Publisher.
9. Sikkh B., & Richard E. (2003). Database Design Using Entity-Relationship Diagrams Auerbach Publications.
10. Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company
11. http://www.basicsofcomputer.com/history_of_database_and_history_of_database_management_system.htm
12. http://www.personal.psu.edu/glh10/ist110/topic/topic07/topic07_06.html
13. http://www.basicsofcomputer.com/history_of_database_and_history_of_database_management_system.htm
14. http://quickbase.intuit.com/articles/timeline-of-database-history
15. http://www.clib.dauniv.ac.in/E-Lecture/ISAR.pdf
16. http://wiki.opflabs.org/display/SP/LSDRT10+Capturing+Representation+Information+from+original+image+files
17. http://en.wikipedia.org/wiki/Information_capture

18. http://en.wikipedia.org/wiki/Information_model
19. http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/postscript/Chapter1.pdf
20. http://ccs1.hnue.edu.vn/hoanpt/DBMS/REF/MBA%20758%20Database%20Management%20System.pdf

Assignment File

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

Presentation Schedule

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

Assessment

There are two aspects to the assessment of this course. First, there are tutor marked assignments; and second, the written examination. Therefore, you are expected to take note of the facts, information and problem solving gathered during the course. The tutor marked assignments must be submitted to your tutor for formal assessment, in accordance to the deadline given. The work submitted will count for 40% of your total course mark.

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

Tutor Marked Assignments (TMAs)

There are TMAs in this course. You need to submit all the TMAs. The best 10 will therefore be counted. When you have completed each assignment, send them to your tutor as soon as possible and make certain that it gets to your tutor on or before the stipulated deadline. If for any reason you cannot complete your assignment on time, contact your tutor before the assignment is due to discuss the possibility of extension. Extension will not be granted after the deadline, unless on extraordinary cases.

Final Examination and Grading

The final examination for CPT 325 will last for a period of 2 hours and have a value of 60% of the total course grade. The examination will consist of questions which reflect the self assessment exercise and tutor marked assignments that you have previously encountered. Furthermore, all areas of the course will be examined. It would be better to use the time between finishing the last unit and sitting for the examination, to revise the entire course. You might find it useful to review your TMAs and comment on them before the examination. The final examination covers information from all parts of the course.

The following are practical strategies for working through this course

1. Read the course guide thoroughly
2. Organize a study schedule. Refer to the course overview for more details. Note the time you are expected to spend on each unit and how the assignment relates to the units. Important details, e.g. details of your tutorials and the date of the first day of the semester are available. You need to gather together all these information in one place such as a diary, a wall chart calendar or an organizer. Whatever method you choose, you should decide on and write in your own dates 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 get behind with their course works. If you get into difficulties with your schedule, please let your tutor know before it is too late for help.
4. Turn to Unit 1 and read the introduction and the objectives for the unit.
5. Assemble the study materials. Information about what you need for a unit is given in the table of content at the beginning of each unit. You will almost always need both the study unit you are working on and one of the materials recommended for further readings, 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 encouraged to read from your set books
7. Keep in mind that you will learn a lot by doing all your assignments carefully. They have been designed to help you meet the objectives of the course and will help you pass the examination.
8. Review the objectives of each study unit to confirm that you have achieved them.

If you are not certain about any of the objectives, review the study material and consult your tutor.

9. When you are confident that you have achieved a unit's objectives, you can start on the next unit. Proceed unit by unit through the course and try to pace your study so that you can keep yourself on schedule.
10. When you have submitted an assignment to your tutor for marking, do not wait for its return before starting on the next unit. Keep to your schedule. When the assignment is returned, pay particular attention to your tutor's comments, both on the tutor marked assignment form and also written on the assignment. Consult you tutor as soon as possible if you have any questions or problems.
11. 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).

Tutors and Tutorials

There are 8 hours of tutorial provided in support of this course. You will be notified of the dates, time and location together with the name and phone number of your tutor as soon as you are allocated a tutorial group. Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties, you might encounter and provide assistance to you during the course. You must mail your tutor marked assignment to your tutor well before the due date. At least two working days are required for this purpose. They will be marked by your tutor and returned to you as soon as possible.

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

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

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

GOODLUCK!

Table of Contents

Module 1

Overview

Unit 1

Introduction to Database Management System (DBMS)

Contents

1.0 Introduction

The success of an organization depends on its ability to acquire accurate and timely data about its operations, to manage this data effectively, and to use it to analyze and guide its activities. In order to carry out the above mentioned function effectively there is need to have a robust Database management system. This unit therefore introduces and traces the history of this very important subject.

2.0 Learning Outcomes

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

i. Define database and database management system.
ii. Discuss the history of database management system
iii. State the advantages and disadvantages of database system
iv. State the functions of database management System

3.0 Learning Contents

3.1 Overview of Database Concepts

Often times, the terms "data" and "information" are often interchangeable in common usage, they denote different concepts. Data consists of facts that represent real-world elements. It is a set of isolated and unrelated raw facts with an implicit meaning.

When the data is processed and converted into a meaningful and useful form, it is known as information. So, information is a collection of facts that have been organized to be valuable to decision makers. Turning data into information is a process or a set of logically related tasks. The ability to use processes to create information is called knowledge.

Character

A character is the most basic element of data that can be observed and manipulated. Behind it are the invisible data elements we call bits and bytes, referring to physical storage elements used by the computer hardware. A character is a single symbol such as a digit, letter, or other special character (e.g., $, #, and?).

Field

A field contains an item of data; that is, a character, or group of characters that are related. For instance, a grouping of related text characters such as "Adepoju Victor" makes up a name in the name field. A field may contain an attribute (e.g., employee salary) or the name of an entity (e.g., person, place, or event).

Record

A record is composed of a group of related fields. One may say that a record contains a collection of attributes related to an entity such as a person or product. The name, address, marital status and telephone number of a single individual would constitute a record. A payroll record would contain the name, address, employee number and title of each employee.

File

A file is defined as a collection of related records. A database file is sometimes called a table. A file may be composed of a complete list of individuals on a mailing list, including their addresses and telephone numbers. Files are frequently categorized by the purpose or application for which they are intended. Some common examples include mailing lists, quality control files, inventory files, or document files. Files may also be classified by the degree of permanence they have. Transition files are only temporary, while master files are much more long-lived.

Database

Database is an organized collection of data, typically describing the activities of one or more related organizations. It can also be defined as a collection of data that exists over a long period of time, often many years managed through a database management system. Database can also be viewed as a collection of related data items. *E.g. Name, Telephone number and Address of your friends. Account number, Name of the account holder and Balance in a banking system, accounts database; payroll database; university's students' database; Amazon's products database; airline reservation database*

A database is similar to a data file in that it is a storage place for data. Like a data file, a database does not present information directly to a user; the user runs an application that accesses data from the database and presents it to the user in an understandable format

Database management System (DBMS)

A ***Database Management System (DBMS)***is software packages designed to store and manage databases. DBMS is software designed to assist in maintaining and utilizing large collections of data, and the need for such systems, as well as their use, is growing rapidly. DBMS provides mechanisms for storing, organizing, retrieving and modifying data for many users. It could be seen as a collection of procedures, languages and programs which can be used to facilitate the design, implementation, operation and maintenance of database systems. DBMS allow for the access and storage of data without concern for the internal representation of data.

A Database system consists of computer software (DBMS), hardware, database administrator (DBA), database administration procedures and the database.

The figure 1 depicts how DBMS manages the interaction between the user and the database.

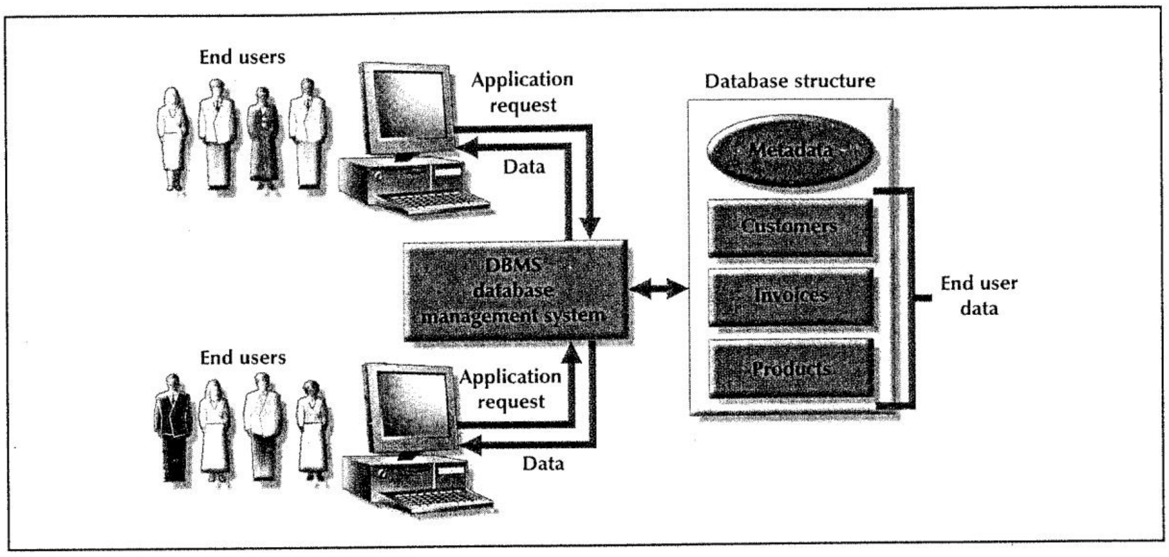


Figure 1: How DBMS manages the Interaction between the user and Database

Self Assessment Exercise(s)

1. Distinguish between data, information, field and record

Self Assessment Answer(s)

3.2 A Timeline of Database History

Ancient Times: Human beings began to store information very long ago. In the ancient times, elaborate database systems were developed by government offices, libraries, hospitals, and business organizations, and some of the basic principles of these systems are still being used today.

1960s: Computerized database started in the 1960s, when the use of computers became a more cost-effective option for private organizations. There were two popular data models in this decade: a network model called CODASYL and a hierarchical model called IMS. One database system that proved to be a commercial success was the SABRE system that was used by IBM to help American Airlines manage its reservations data.

1970 to 1972: E.F. Codd published an important paper to propose the use of a relational database model, and his ideas changed the way people thought about databases. In his model, the database's schema, or logical organization, is disconnected from physical information storage, and this became the standard principle for database systems.

1970s: Two major relational database system prototypes were created between the years 1974 and 1977, and they were the Ingres, which was developed at UBC, and System R, created at IBM San Jose. Ingres used a query language known as QUEL, and it led to the creation of systems such as Ingres Corp., MS SQL Server, Sybase, Wang's PACE, and Britton-Lee. On the other hand, System R used the SEQUEL query language, and it contributed to the development of SQL/DS, DB2, Allbase, Oracle, and Non-Stop SQL. It was also in this decade that Relational Database Management System, or RDBMS, became a recognized term.

1976: A new database model called Entity-Relationship, or ER, was proposed by P. Chen this year. This model made it possible for designers to focus on data application, instead of logical table structure.

1980s: Structured Query Language, or SQL, became the standard query language.

Relational database systems became a commercial success as the rapid increase in computer sales boosted the database market, and this caused a major decline in the popularity of network and hierarchical database models. DB2 became the flagship database product for IBM, and the introduction of the IBM PC resulted in the establishments of many new database companies and the development of products such as PARADOX, RBASE 5000, RIM, Dbase III and IV, OS/2 Database Manager, and Watcom SQL.

Early 1990s: After a database industry shakeout, most of the surviving companies sold complex database products at high prices. Around this time, new client tools for application development were released, and these included the Oracle Developer, PowerBuilder, VB, and others. A number of tools for personal productivity, such as ODBC and Excel/Access, were also developed. Prototypes for Object Database Management Systems, or ODBMS, were created in the early 1990s.

Mid 1990s: The advent of the Internet led to exponential growth of the database industry. Average desktop users began to use client-server database systems to access computer systems that contained legacy data.

Late 1990s: Increased investment in online businesses resulted in a rise in demand for Internet database connectors, such as Front Page, Active Server Pages, Java Servelets, Dream Weaver, ColdFusion, Enterprise Java Beans, and Oracle Developer 2000. The use of cgi, gcc, MySQL, Apache, and other systems brought open source solution to the Internet. With the increased use of point-of-sale technology, online transaction processing and online analytic processing began to come of age.

2000s: Although the Internet industry experienced a decline in the early 2000s, database applications continue to grow. New interactive applications were developed for PDAs, point-of-sale transactions, and consolidation of vendors. Presently, the three leading database companies in the western world are Microsoft, IBM, and Oracle.

Self Assessment Exercise(s)

1. Who proposed FR Model?
2. When did SQL become standard language?

Self Assessment Answer(s)

3.3 Database System Applications

Databases are widely used in some of the representative applications listed below:

1. ***Banking***: For customer information, accounts, and loans, and banking transactions.
2. ***Airlines***: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner—terminals situated around the world accessed the central database system through phone lines and other data networks.
3. ***Universities***: For student information, course registrations, and grades.
4. ***Credit card transactions***: For purchases on credit cards and generation of monthly statements.
5. ***Telecommunication***: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
6. ***Finance***: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
7. ***Sales***: For customer, product, and purchase information.
8. ***Manufacturing***: For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items.
9. ***Human resources***: For information about employees, salaries, payroll taxes and benefits, and for generation of paychecks.

As the list illustrates, databases form an essential part of almost all enterprises today

Self Assessment Exercise(s)

1. Mention five areas where database can be applied?

Self Assessment Answer(s)

3.4 Advantages of DBMS

Using a DBMS to manage data has many advantages

1. **Data independence:** Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.
2. **Efficient data access:** A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.
3. **Data integrity and security:** If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce *access controls* that govern what data is visible to different classes of users.
4. **Data administration:** When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals, who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimizeredundancy and for fine-tuning the storage of the data to make retrieval efficient.
5. **Concurrent access and crash recovery:** A DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.
6. **Reduced application development time:** Clearly, the DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of applications. Such applications are also likely to be more robust than applications developed from scratch because many important tasks are handled by the DBMS instead of being implemented by the application

Disadvantages of a DBMS

There are basically two major downsides to using DBMS. One of these is cost, and the other the threat to data security.

Cost: Implementing a DBMS system can be expensive and time-consuming, specially in large organizations. Training requirements alone can be quite costly.

Security: Even with safeguards in place, it may be possible for some unauthorized users to access the database. In general, database access is an all or nothing proposition. Once an unauthorized user gets into the database, they have access to all the files, not just a few. Depending on the nature of the data involved, these breaches in security can also pose a threat to individual privacy. Steps should also be taken to regularly make backup copies of the database files and store them because of the possibility of fires and earthquakes that might destroy the system.

Other disadvantaged includes

Large size – database occupies lots of hard disk space and RAM

Complexity – DBMS is hard to learn; wrong choices may spell irreversible disaster later.

Greater impact of failure – failure impacts *all* users.

More difficult recovery –In case it crashes the recovery is often difficulty since it is more complex.

Self Assessment Exercise(s)

1. Highlight the advantages and disadvantages of Database Management System?

Self Assessment Answer(s)

3.5 Functions of Database Management System

The functions performed by a typical DBMS are the following:

Data Definition: The DBMS provides functions to define the structure of the data in the application. These include defining and modifying the record structure, the type and size of fields and the various constraints/conditions to be satisfied by the data in each field.

Data Manipulation Once the data structure is defined, data needs to be inserted, modified or deleted. The functions which perform these operations are also part of the DBMS. These functions can handle planned and unplanned data manipulation needs. Planned queries are those which form part of the application. Unplanned queries are ad-hoc queries which are performed on a need basis.

Data Security & Integrity: The DBMS contains functions which handle the security and integrity of data in the application. These can be easily invoked by the application and hence the application programmer need not code these functions in his/her programs.

Data Recovery & Concurrency: Recovery of data after a system failure and concurrent access of records by multiple users are also handled by the DBMS.

Data Dictionary Maintenance: Maintaining the Data Dictionary which contains the data definition of the application is also one of the functions of a DBMS.

Performance: Optimizing the performance of the queries is one of the important functions of a DBMS. Hence the DBMS has a set of programs forming the Query Optimizer which evaluates the different implementations of a query and chooses the best among them.

Thus, the DBMS provides an environment that is both convenient and efficient to use when there is a large volume of data and many transactions to be processed.

4.0 Conclusion

In this unit you have learnt the overview of database concept which are terms like data, information, record, file, database, database system, database management system. You also learnt the history, advantages, disadvantages and functions of database management system.

5.0 Summary

You have learnt that:

i. Data consists of facts that represent real-world elements.

ii. Information is a collection of facts that have been organized to be valuable to decision makers.

iii. A character is the most basic element of data that can be observed and manipulated

iv. A field contains an item of data; that is, a character, or group of characters that are related.

v. A record is composed of a group of related fields.

vi. A file is defined as a collection of related records. A database file is sometimes called a table.

vii. Database is an organized collection of data, typically describing the activities of one or more related organizations.

viii. DBMS provides mechanisms for storing, organizing, retrieving and modifying data for many users

ix. DBMS can be used in Banking, Airlines, Universities, Credit card transactions: Telecommunication: Finance, Sales, Manufacturing etc

6.0 Tutor-Marked Assignment

i. Distinguish between database and database management system
ii. Discuss some of the areas for database applications

7.0 References/Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Connolly, T., Begg, C., and Strachan, A.(1998) *Database Systems, A Practical Approach to Design, Implementation, and Management*, Addison- Wesley, Harlow, England.

Earp, R. and Bagui, S. (2000) *"Building an Entity Relationship Diagram: A Software Engineering Approach," Database Management*, Auerbach Publications, Boca Raton, FL, 22-10-41, 1–16, Elmasri, R. and Navathe, S.B. (2000) *Fundamentals of Database Systems*, 3rd ed., Addison-Wesley, Reading, MA.

Fred R. M., Jeffrey A.H., Mary B. P. (2005). Modern Database Management (7th ed.), Prentice Hall.

http://www.basicsofcomputer.com/history_of_database_and_history_of_database_management_system.htm

http://www.personal.psu.edu/glh10/ist110/topic/topic07/topic07_06.html

http://www.basicsofcomputer.com/history_of_database_and_history_of_database_management_system.htm

http://quickbase.intuit.com/articles/timeline-of-database-history

http://quickbase.intuit.com/articles/timeline-of-database-history

Unit 2

Classification of Database Management System

Contents

1.0 Introduction

DBMSs come in many shapes and sizes. Since there are a variety of DBMSs available, you should know some of the basic features, as well as strengths and weaknesses of the major types. This unit takes a comprehensive look at the various classification of DBMS based on Model, number of user, number of sites and on purpose.

2.0 Learning Outcomes

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

i. Classify Database management system based on model, users, sites and purpose.
ii. Explain each of the classifications above
iii. State the advantages and disadvantages of each classification above

3.0 Learning Contents

3.1 Classification Based on Data Model

There are 4 different types of DBMS based on the model they use. These are based upon their management of database structures. In other words, the types of DBMS are entirely dependent upon how the database is structured by that particular DBMS. They are discussed below,

Hierarchical Databases

Hierarchical Databases (DBMS), commonly used on mainframe computers, have been around for a long time. It is one of the oldest methods of organizing and storing data and it is still used by some organizations for making travel reservations. A hierarchical database is organized in pyramid fashion, like the branches of a tree extending downwards. Related fields or records are grouped together so that there are higher-level records and lower-level records, just like the parents in a family tree sit above the subordinated children.

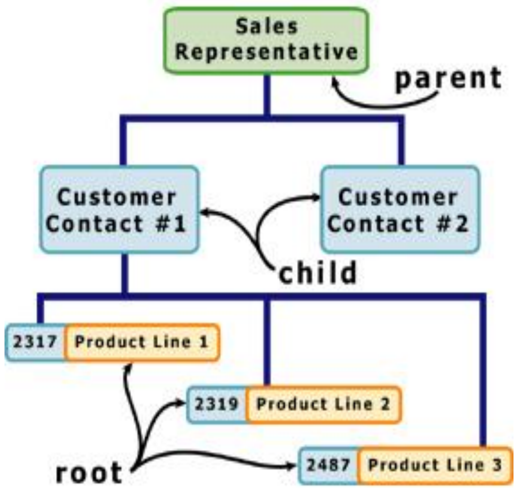


Figure 2.1: Hierarchical database

Based on this analogy, the parent record at the top of the pyramid is called the **root record**. A child record always has only one parent record to which it is linked, just like in a normal family tree. In contrast, a parent record may have more than one child record linked to it. Hierarchical databases work by moving from the top down. A record search is conducted by starting at the top of the pyramid and working down through the tree from parent to child until the appropriate child record is found. Furthermore, each child can also be a parent with children underneath it.

The advantage of hierarchical databases is that they can be accessed and updated rapidly because the tree-like structure and the relationships between records are defined in advance. However, this feature is a two-edged sword. The disadvantage of this type of database structure is that each child in the tree may have only one parent, and relationships or linkages between children are not permitted, even if they make sense from a logical standpoint. Hierarchical databases are so rigid in their design that adding a new field or record requires that the entire database be redefined.

Network Databases

Network databases are similar to hierarchical databases by also having a hierarchical structure. There are a few key differences, however. Instead of looking like an upside-down tree, a network database looks more like a cobweb or interconnected network of records. In network databases, children are called **members** and parents are called **owners**. The most important difference is that each child or member can have more than one parent (or owner).

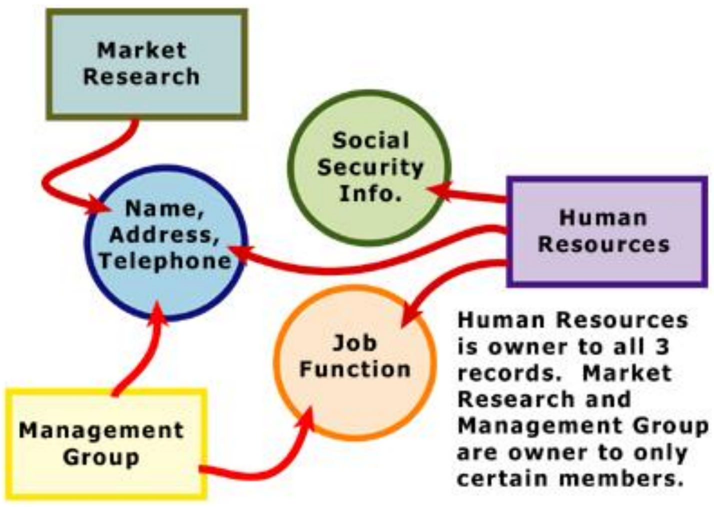
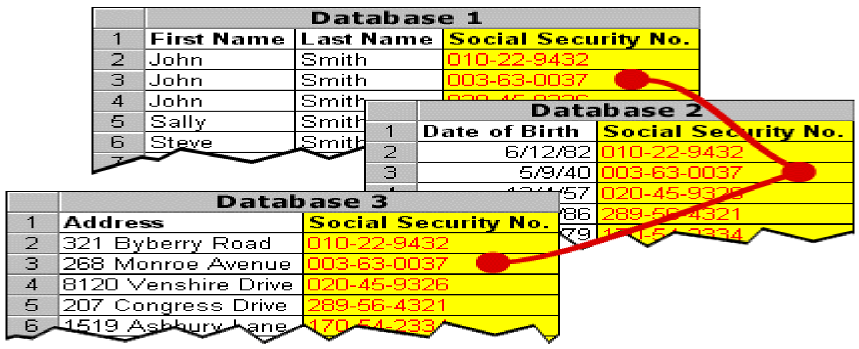

Market Research
Social Security Info.
Name, Address, Telephone
Human Resources
Job Function
Management Group
Human Resources is owner to all 3 records. Market Research and Management Group are owner to only certain members.

Figure 2.2: Network database

Like hierarchical databases, network databases are principally used on mainframe computers. Since more connections can be made between different types of data, network databases are considered more flexible. Thus the structure of a network database is extremely complicated because of many-to-many relationships in which one record can be used as a key of the entire database. However, two limitations must be considered when using this kind of database. Similar to hierarchical databases, network databases must be defined in advance. There is also a limit to the number of connections that can be made between records.

Relational Databases

In relational databases, the relationship between data files is relational, not hierarchical. Hierarchical and network databases require the user to pass down through a hierarchy in order to access needed data. Relational databases connect data in different files by using common data elements or a key field. Data in relational databases is stored in different tables, each having a key field that uniquely identifies each row. Relational databases are more flexible than either the hierarchical or network database structures. In relational databases, tables or files filled with data are called **relations**, **tuples** designate a row or record, and columns are referred to as **attributes** or fields.



Database 1

1	First Name	Last Name	Social Security No.
2	John	Smith	010-22-9432
3	John	Smith	003-63-0037
4	John	Smith	020-45-9326
5	Sally	Smith	
6	Steve	Smith	

Database 2

1	Date of Birth	Social Security No.
2	6/12/82	010-22-9432
3	5/9/40	003-63-0037
4	/57	020-45-9326
5	/86	289-56-4321
6	79	170-54-2334

Database 3

1	Address	Social Security No.
2	321 Byberry Road	010-22-9432
3	268 Monroe Avenue	003-63-0037
4	8120 Venshire Drive	020-45-9326
5	207 Congress Drive	289-56-4321
6	1519 Ashbury Lane	170-54-2334

Figure 2.3: Relational database

Relational databases work on the principle that each table has a key field that uniquely identifies each row, and that these key fields can be used to connect one table of data to another. Thus, one table might have a row consisting of a customer account number as the key field along with address and telephone number. The customer account number in this table could be linked to another table of data that also includes customer account number (a key field), but in this case, contains information about product returns, including an item number (another key field). This key field can be linked to another table that contains item numbers and other product information such as production location, color, quality control person, and other data. Therefore, using this database, customer information can be linked to specific product information.

The relational database has become quite popular for two major reasons. First, relational databases can be used with little or no training. Second, database entries can be modified without redefining the entire structure. The downside of using a relational database is that searching for data can take more time than if other methods are used.

A number of RDBMSs are available; some popular examples are Oracle, Sybase, Ingress, Informix, Microsoft SQL Server, and Microsoft Access.

Object-oriented Databases (OODBMS)

This can handle many new data types, including graphics, photographs, audio, and video, **object-oriented databases** represent a significant advance over their other database cousins. Hierarchical and network databases are all designed to handle structured data; that is, data that fits nicely into fields, rows, and columns. They are useful for handling small snippets of information such as names, addresses, zip codes, product numbers, and any kind of statistic or number you can think of. On the other hand, an object-oriented database can be used to store data from a variety of media sources, such as photographs and text, and produce work, as output, in a multimedia format.

Object-oriented databases use small, reusable chunks of software called objects. The objects themselves are stored in the object-oriented database. Each object consists of two elements:

1) a piece of data (e.g., sound, video, text, or graphics), and

2) the instructions, or software programs called methods, for what to do with the data.

The instructions contained within the object are used to do something with the data in the object. For example, test scores would be within the object as would the instructions for calculating average test score.

Object-oriented databases have two disadvantages. First, they are more costly to develop. Second, most organizations are reluctant to abandon or convert from those databases that they have already invested money in developing and implementing. However, the benefits to object-oriented databases are compelling. The ability to mix and match reusable objects provides incredible multimedia capability. Healthcare organizations, for example, can store, track, and recall CAT scans, X-rays, electrocardiograms and many other forms of crucial data

Object-Relational Database

An object-relational database (ORD) or object-relational database management system (ORDBMS) is a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, it supports extension of the data model with custom data-types and methods. One aim for this type of system is to bridge the gap between conceptual data modeling techniques such as Entity-relationship diagram (ERD) and object-relational mapping (ORM), which often use classes and inheritance, and relational databases, which do not directly support them.

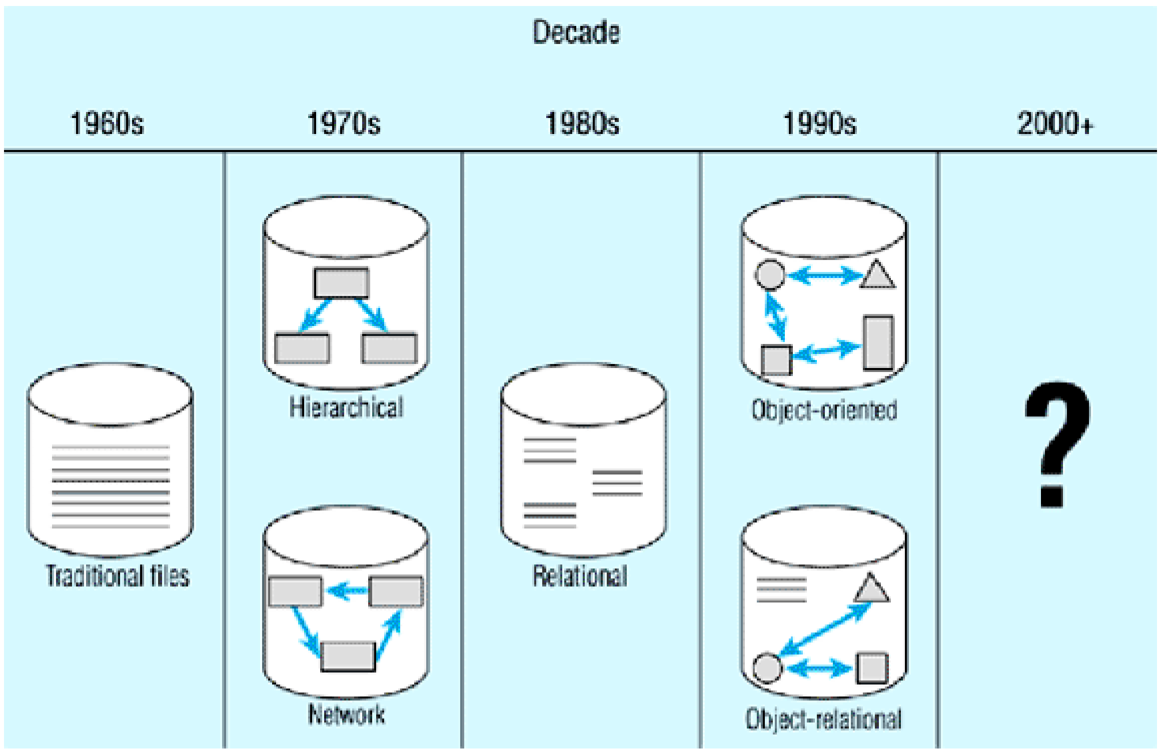
Another related, aim is to bridge the gap between relational databases and the object-oriented modeling techniques used in programming languages such as Java, C++ or C# However, a more popular alternative for achieving such a bridge is to use a standard relational database system with some form of ORM software.

Whereas traditional RDBMS or SQL-DBMS products focused on the efficient management of data drawn from a limited set of data-types (defined by the relevant language standards), an object-relational DBMS allows software-developers to integrate their own types and the methods that apply to them into the DBMS. ORDBMS technology aims to allow developers to raise the level of abstraction at which they view the problem domain. This goal is not universally shared; proponents of relational databases often argue that object-oriented specification *lowers* the abstraction level.

An object-relational database can be said to provide a middle ground between relational databases and *object-oriented databases* (OODBMS). In object-relational databases, the approach is essentially that of relational databases: the data resides in the database and is manipulated collectively with queries in a query language; at the other extreme are OODBMSes in which the database is essentially a persistent object store for software written in an object-oriented programming language, with a programming API for storing and retrieving objects, and little or no specific support for querying.

Many SQL ORDBMSs on the market today are extensible with user defined types (UDT) and custom-written functions (e.g. stored procedures. Some (e.g. SQL Server) allow such functions to be written in object-oriented programming languages, but this by itself doesn't make them object-oriented databases; in an object-oriented database, object orientation is a feature of the data model. Figure 2.4 shows the trend.

Figure 2.4. Trends of database



Self Assessment Exercise(s)

1. Attempt a classification of Database Management System by Model.

Self Assessment Answer(s)

3.2 Classification Based on Number of Users

Depending on the number of users the DBMS supports, it is divided into two categories, namely, *single-user system* and *multi-user system*. In single-user system the database resides on one computer and is only accessed by one user at a time. The user may design, maintain, and write programs for accessing and manipulating the database according to the requirements, as well as perform all the user roles. The user may also hire database system designers to design a system. In such a case, the single user performs the role of end user only.

However, in most enterprises the large amount of data is to be managed and accessed by multiple users and thus, requires multi-user systems. In multi-user system, multiple users can access the database simultaneously. In multi-user DBMS, the data is both integrated and shared. For example, the *Online Book* database is a multi-user database system in which the data of books, authors, and publishers are stored centrally and can be accessed by many users.

3.3 Classification Based on Number of Sites

Depending on the number of sites over which the database is distributed, it is divided into two types, namely, *centralized* and *distributed database systems*.

Centralized database systems run on a single computer system. Both the database and DBMS software reside at a single computer site. The user interacts with the centralized system through a dummy terminal connected to it for information retrieval.

In distributed database systems, the database and DBMS are distributed over several computers located at different sites. The computers communicate with each other through various communication media such as high-speed networks or telephone lines. Distributed databases can be classified as *homogeneous* and *heterogeneous*. In **homogeneous** distributed database system, all sites have identical database management system software, whereas in **heterogeneous** distributed database system, different sites use different database management system software.

3.4 Classification Based on the Purpose

Depending on the purpose the DBMS serves, it can be classified as **general purpose** or **specific purpose**. DBMS is a general-purpose software system. It can; however, be designed for specific purposes such as airline or railway reservation. Such systems cannot be used for other applications without major changes. These database systems fall under the category of Online Transaction Processing (OLTP) systems.

Online transaction processing system is specifically used for data entry and retrieval. It supports large number of concurrent transactions without excessive delays. An automatic teller machine (ATM) for a bank is an example of online commercial transaction processing application. The OLTP technology is used in various industries, such as banking, airlines, supermarkets, manufacturing, etc.

Self Assessment Exercise(s)

1. Distributed Database can be classified as?
2. Classification based on users can be

Self Assessment Answer(s)

4.0 Conclusion

In this unit you studied the various ways database can be classified according to model, users, sites and purpose.

5.0 Summary

1. Relational databases connect data in different files by using common data elements or a key field. Data in relational databases is stored in different tables, each having a key field that uniquely identifies each row
2. In a hierarchical model, data is organized into an inverted treelike structure, implying a multiple downward link in each node to describe the nesting, and a sort field to keep the records in a particular order in each same-level list.
3. In the network model, records can participate in any number of named relationships. Each relationship associates a record of one type (called the owner) with multiple records of another type (called the member).
4. An object-relational database (ORD) or object-relational database management system (ORDBMS) is a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language.
5. In an object database (also object oriented database), information is represented in the form of objects as used in object-oriented programming.
6. Database can also be single user or multi user based on number of users
7. Database can be centralized or distributed based on site
8. Database can be as general purpose or specific purpose based on purpose

6.0 Tutor Marked Assignment

1 Distinguish between various model of DBMS based on model

2 Give taxonomy of database management systems

7.0 References/Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Right R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://www.basicsofcomputer.com/history_of_database_and_history_of_database_management_system.htm

http://www.personal.psu.edu/glh10/ist110/topic/topic07/topic07_06.html

http://www.basicsofcomputer.com/history_of_database_and_history_of_database_management_system.htm

http://quickbase.intuit.com/articles/timeline-of-database-history

Unit 3

Component of a Database System

Contents

1.0 Introduction

This unit discusses the various components of a DBMS which include users, hardware, software and data. All these components interrelate so as to have an efficient and effective system.

2.0 Learning Outcomes

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

i. Identify the various components of a DBMS
ii. State the interrelationship among the various components,
iii. Describe the various users of a DBMS
iv. State the functions of database administrator

3.0 Learning Contents

3.1 Components of Database

A database system is composed of four components; Data, Hardware, Software and Users which coordinate with each other to form an effective database system.

USERS:

Users are those persons who need the information from the database to carry out their primary business responsibilities i.e. Personnel, Staff, Clerical, Managers, Executives etc. On the basis of the job and requirements made by them they are provided access to the database totally or partially. Database users are those who interact with the database in order to query and update the database and generate reports. Database users are further classified into the following categories:

a. Naive users – are unsophisticated users who interact with the system by invoking one of the permanent application programs that have been written previously.

b. Application programmers – are computer professionals who write application programs. They interact with system through Data Manipulation Language (DML) and Data Definition Language (DFL) calls

c. Sophisticated users – interact with the system without writing programs. Instead they form their requests in a database query language.

d. Specialized users – are sophisticated users who write specialized database applications that do not fit into the traditional data processing framework.

Database Administrators (DBA) is responsible for the design, construction, and maintenance of a database. The DBA is responsible for many critical tasks which include:

i. **Design of the conceptual and physical schemas:** The DBA is responsible for interacting with the users of the system to understand what data is to be stored in the DBMS and how it is likely to be used. Based on this knowledge, the DBA must design the conceptual schema (decide what relations to store) and the physical schema (decide how to store them). The DBA may also design widely used portions of the external schema, although users will probably augment this schema by creating additional views.

ii. **Security and authorization:** The DBA is responsible for ensuring that unauthorized data access is not permitted. In general, not everyone should be able to access all the data. In a relational DBMS, users can be granted permission to access only certain views and relations. For example, although you might allow students to find out course enrollments and who teaches a given course, you would not want students to see faculty salaries or each others' grade information. The DBA can enforce this policy by giving students permission to read only the Courseinfo view.
iii. **Data availability and recovery from failures:** The DBA must take steps to ensure that if the system fails, users can continue to access as much of the uncorrupted data as possible. The DBA must also work to restore the data to a consistent state. The DBMS provides software support for these functions, but the
iv. DBA is responsible for implementing procedures to back up the data periodically and to maintain logs of system activity (to facilitate recovery from a crash).
v. **Database tuning:** The needs of users are likely to evolve with time. The DBA is responsible for modifying the database, in particular the conceptual and physical schemas, to ensure adequate performance as user requirements change

Software:

The Software part consists of DBMS which acts as a bridge between the user and the database or in other words, software that interacts with the users, application programs, and database and files system of a particular storage media (hard disk, magnetic tapes etc.) to insert, update, delete and retrieve data. For performing these operations such as insertion, deletion and updating we can either use the Query Languages like SQL, QUEL, Gupta SQL or application software such as Visual Basic, Developer etc. Software

i. Controls the organization, storage, management, and retrieval of data in a database.
ii. includes operating system, network software, and the application programs
iii. Encompasses the physical interconnections and devices required to store and execute (or run) the software.
iv. consists of a machine language specific to an individual processor.
v. is usually written in high-level programming language more efficient for humans to use

Hardware:

The hardware consists of the secondary storage devices such as magnetic disks (hard disk, zip disk, floppy disks), optical disks (CD-ROM), magnetic tapes etc. on which data is stored together with the Input/Output devices (mouse, keyboard, printers), processors, main memory etc. which are used for storing and retrieving the data in a fast and efficient manner. Since database can range from those of a single user with a desktop computer to those on mainframe computers with thousands of users, therefore proper care should be taken for choosing appropriate hardware devices for a required database.

i. Hardware of a system can range from a PC to a network of computers.
ii. It also includes various storage devices like hard discs and input and output devices like monitor, printer, etc.

Data:

It is a very important component of the database system. Most of the organizations generate, store and process 1arge amount of data. The data acts a bridge between the machine parts i.e. hardware and software and the users which directly access it or access it through some application programs. Data stored in a database includes numerical data such as whole numbers and floating-point numbers and non-numerical data such as characters, date, or logical data. More advanced systems may include more complicated data entities such as pictures and images as data types. It can be

User Data: It consists of a table(s) of data called Relation(s) where Column(s) are called fields of attributes and rows are called Records for tables. A Relation must be structured properly.

Metadata: - A description of the structure of the database is known as Metadata. It basically means "data about data". System Tables store the Metadata which includes.

- Number of Tables and Table Names

- Number of fields and field Names

- Primary Key Fields

Application Metadata - It stores the structure and format of Queries, reports and other applications components.

Self Assessment Exercise(s)

1. What are the various components of DBMS
2. Discuss the functions of a Database Administrator.

Self Assessment Answer(s)

4.0 Conclusion

In this unit you studied the various component of a DBMS which are users, software, hardware and data components.

5.0 Summary

You have learnt that:

1. **Naive users –** are unsophisticated users who interact with the system by invoking one of the permanent application programs that have been written previously.
2. **Application programmers –** are computer professionals who write application programs.
3. **Sophisticated users –** interact with the system without writing programs. Instead, they form their requests in a database query language.
4. **Specialized users –** are sophisticated users who write specialized database applications that do not fit into the traditional data processing framework.
5. DBA is responsible for the design, construction, and maintenance of a database. He/she is also responsible for Design of the conceptual and physical schemas, Security and authorization, Data availability and recovery from failures etc
6. Hardware components consist of their devices connected to the computer. Hardware of a system can range from a PC to a network of computers. It also includes various storage devices like hard discs and input and output devices like monitor, printer, etc
7. Data is a very important component of the DBMS and acts a bridge between the machine parts i.e. hardware and software and the users which directly access it or access it through some application programs
8. The Software part consists of DBMS which acts as a bridge between the user and the database

6.0 Tutor Marked Assignment

1. Distinguish between different types of users of a DBMS
2. Discuss the components of a DBMS

7.0 References/Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://www.basicsofcomputer.com/history_of_database_and_history_of_database_management_system.htm

http://www.personal.psu.edu/glh10/ist110/topic/topic07/topic07_06.html http://www.basicsofcomputer.com/history_of_database_and_history_of_database_management_system.htm

http://quickbase.intuit.com/articles/timeline-of-database-history

http://ccs1.hnue.edu.vn/hoanpt/DBMS/REF/MBA%20758%20Database%20Management%20System.pdf

http://deden08m.files.wordpress.com/2011/03/ch06-database-management-systems.pdf

Unit 4

DBMS Architecture and Data Independence

1.0 Introduction

This unit discusses the architecture of DBMS as well as data independence.

2.0 Learning Outcomes

At the end of this unit you should:

i. Explain DBMS architecture
ii. Explain the concept of conceptual design
iii. Discuss physical design and view design
iv. Explain the concept of data independence

3.0 Learning Contents

3.1 DBMS Architecture

The DBMS architecture describes how data in the database is viewed by the users. It is not concerned with how the data is handled and processed by the DBMS. The database users are provided with an abstract view of the data by hiding certain details of how data is physically stored. This enables the users to manipulate the data without worrying about where it is located or how it is actually stored.

In this architecture, the overall database description can be defined at three levels, namely, *internal, conceptual*, and *external* levels and thus, named **three-level DBMS architecture**. This architecture is proposed by ANSI/SPARC (American National Standards Institute/Standards Planning and Requirements Committee) and hence, is also known as **ANSI/SPARC architecture**. The three levels are discussed here.

i. **Internal level**: It is the lowest level of data abstraction that deals with the physical representation of the database on the computer and thus, is also known as **physical level**. It describes *how* the data is physically stored and organized on the storage medium. At this level, various aspects are considered to achieve optimal runtime performance and storage space utilization. These aspects include storage space allocation techniques for data and indexes, access paths such as indexes, data compression and encryption techniques, and record placement.

ii. **Conceptual level**: This level of abstraction deals with the logical structure of the entire database and thus, is also known as **logical level**. It describes *what* data is stored in the database, the relationships among the data and complete view of the user’s requirements without any concern for the physical implementation. That is, it hides the complexity of physical storage structures. The conceptual view is the overall view of the database and it includes all the information that is going to be represented in the database.

iii. **External level**: It is the highest level of abstraction that deals with the user’s view of the database and thus, is also known as **view level**. In general, most of the users and application programs do not require the entire data stored in the database. The external level describes a part of the database for a particular group of users. It permits users to access data in a way that is customized according to their needs, so that the same data can be seen by different users in different ways, at the same time. In this way, it provides a powerful and flexible security mechanism by hiding the parts of the database from certain users, as the user is not aware of existence of any attributes that are missing from the view.

These three levels are used to describe the schema of the database at various levels. Thus, the three-level architecture is also known as **three-schema architecture**. The internal level has an **internal schema**, which describes the physical storage structure of the database. The conceptual level has a **conceptual schema**, which describes the structure of entire database. The external level has **external schemas** or **user views**, which describe the part of the database according to a particular user's requirements and hide the rest of the database from that user. The physical level is managed by the operating system under the direction of DBMS. The three-schema architecture is shown in figure 1.

To understand the three-schema architecture, consider the three levels of the BOOK file in *Online Book* database as shown in figure 2. In this figure, two views (*view 1* and *view 2*) of the BOOK file have been defined at the external level. Different database users can see these views. The details of the data types are hidden from the users. At the conceptual level, the BOOK records are described by a type definition. The application programmers and the DBA generally work at this level of abstraction. At the internal level, the BOOK records are described as a block of consecutive storage locations such as words or bytes. The database users and the application programmers are not aware of these details; however, the DBA may be aware of certain details of the physical organization of the data.

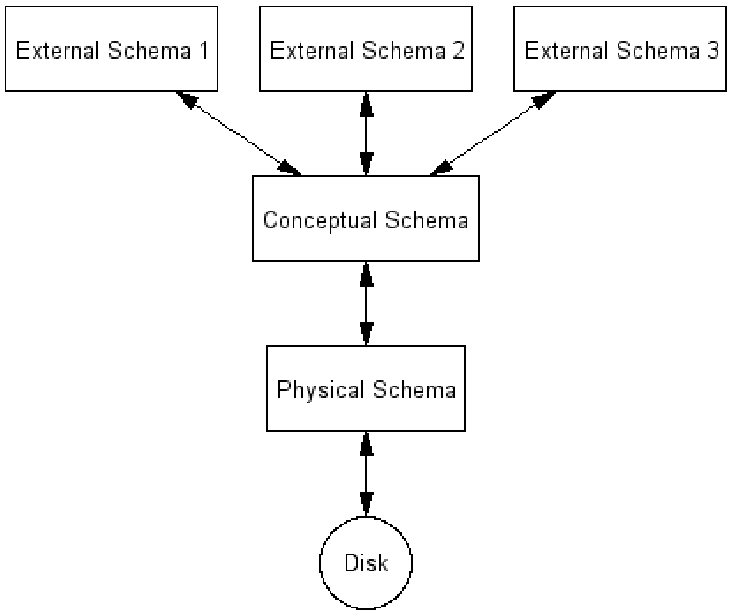


Fig. 4.1 Three-schema architecture

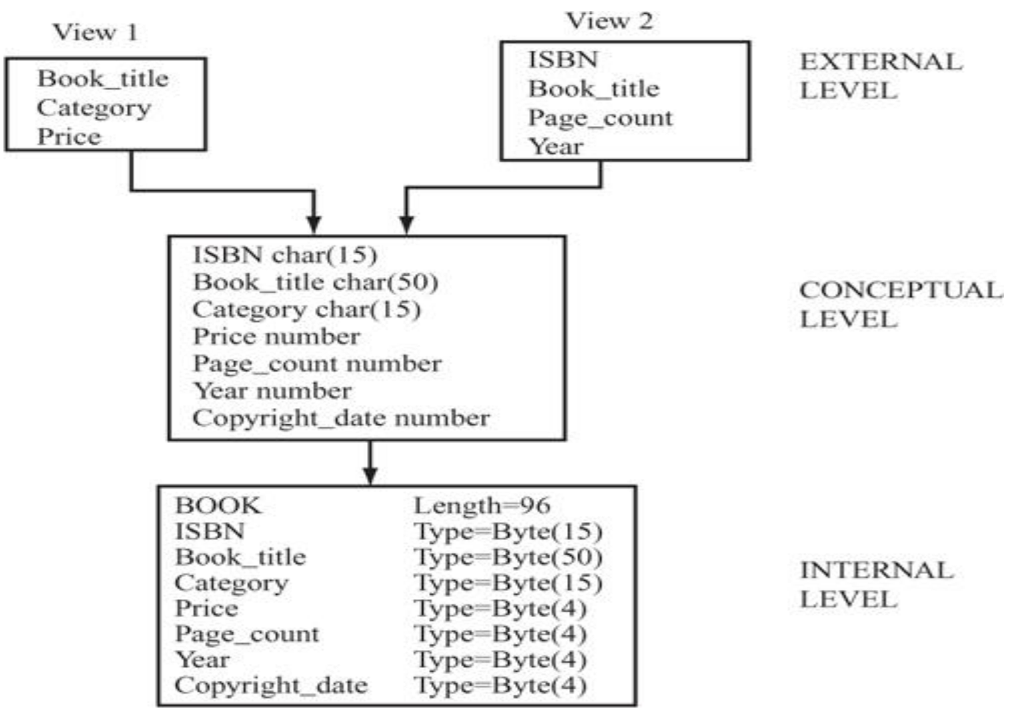


Fig.4.2. Three levels of Online Book database (BOOK file)

In three-schema architecture, each user group refers only to its own external view. Whenever a user specifies a request to generate a new external view, the DBMS must transform the request specified at external level into a request at conceptual level, and then into a request at physical level. If the user requests for data retrieval, the data extracted from the database must be presented according to the need of the user. This process of transforming the requests and results between various levels of DBMS architecture is known as **mapping**.

Objectives of Three Level Architecture

The database views were suggested because of following reasons or objectives of levels of a database:

1. Make the changes easy in database when some changes needed by environment.
2. The external view or user views do not depend upon any change made ii other view. For example, changes in hardware, operating system or internal view should not change the external view.
3. The users of database should not worry about the physical implementation and internal working of database system.
4. The data should reside at same place and all the users can access it as per their requirements.
5. DBA can change the internal structure without affecting the user's view.
6. The database should be simple and changes can be easily made.
7. It is independent of all hardware and software.

3.2 Data Independence

The main advantage of three-schema architecture is that it provides data independence. Data independence is the ability to change the schema at one level of the database system without having to change the schema at the other levels. Data independence is of two types, namely, *logical data independence* and *physical data independence*.

i. **Logical data independence**: It is the ability to change the conceptual schema without affecting the external schemas or application programs. The conceptual schema may be changed due to change in constraints or addition of new data item or removal of existing data item, etc., from the database. The separation of the external level from the conceptual level enables the users to make changes at the conceptual level without affecting the external level or the application programs. For example, if a new data item, say Edition is added to the BOOK file, the two views (*view 1* and *view 2* shown in Figure2) are not affected.

ii. **Physical data independence**: It is the ability to change the internal schema without affecting the conceptual or external schema. An internal schema may be changed due to several reasons such as for creating additional access structure, changing the storage structure, etc. The separation of internal schema from the conceptual schema facilitates physical data independence.

Logical data independence is more difficult to achieve than the physical data independence because the application programs are always dependent on the logical structure of the database. Therefore, the change in the logical structure of the database may require change in the application programs.

Self Assessment Exercise(s)

1. What is the main advantage of three schema architecture?
2. Which level of abstraction deals with the logical structure of the entire database?
3. The highest level of abstraction is?

Self Assessment Answer(s)

4.0 Conclusion

In this unit you have learnt the architecture of database management system as well as data independence

5.0 Summary

You have learnt:

1. The three-level architecture is also known as **three-schema architecture**.
2. The internal level has an **internal schema**, which describes the physical storage structure of the database.
3. The conceptual level has a **conceptual schema**, which describes the structure of entire database.
4. The external level has **external schemas** or **user views**, which describe the part of the database according to a particular user's requirements and hide the rest of the database from that user. The physical level is managed by the operating system under the direction of DBMS
5. Logical data independence is the ability to change the conceptual schema without affecting the external schemas or application programs. While Physical data independence: It is the ability to change the internal schema without affecting the conceptual or external schema

6.0 Tutor-Marked Assignment

Explain the difference between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence?

7.0 References/Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://www.nou.edu.ng/noun/NOUN_OCL/pdf/pdf2/CIT%20744.pdf

http://bitcse.selfip.com/LECTURE%20NOTES/DBMS_SN/1.Overview.pdf

http://my.safaribooksonline.com/book/databases/9788131731925/database-system/ch01lev1sec8

http://searchsqlserver.techtarget.com/definition/relational-database

Module 2

Database Model

Unit 1:	Entity Relationship (ER) Model
Unit 2:	Relational Database
Unit 3:	Functional Dependency & Database Normalisation

Unit 1

Entity Relationship Model

Content

1.0 Introduction

This unit provides an overview of the steps involves in database design process and Entity Relationship Model

2.0 Learning Outcomes

At the end of the unit you should able to:

i. Describe the steps in designing a database
ii. Explain ER model
iii. Defied entity, attributes, relationship
iv. Explain cardinality and constraints

3.0 Learning Contents

3.1 Overview of Database Design

The database design process can be divided into six steps. The ER model is most relevant to the first three steps:

(1) Requirements Analysis: The very first step in designing a database application is to understand what data is to be stored in the database, what applications must be built on top of it, and what operations are most frequent and subject to performance requirements. In other words, we must find out what the users want from the database

This is usually an informal process that involves discussions with user groups, a study of the current operating environment and how it is expected to change, analysis of any available documentation on existing applications that are expected to be replaced or complemented by the database, and so on. Several methodologies have been proposed for organizing and presenting the information gathered in this step, and some automated tools have been developed to support this process.

(2) Conceptual Database Design: The information gathered in the requirements analysis step is used to develop a high-level description of the data to be stored in the database, along with the constraints that are known to hold over this data. This step is often carried out using the ER model, or a similar high-level data model, and is discussed in the rest of this unit.

(3) Logical Database Design: We must choose a DBMS to implement our database design and convert the conceptual database design into a database schema in the data model of the chosen DBMS. We will only consider relational DBMSs, and therefore, the task in the logical design step is to convert an ER schema into a relational database schema. The result is a conceptual schema, sometimes called the **logical schema**, in the relational data model

(4) Schema Refinement: The fourth step in database design is to analyze the collection of relations in our relational database schema to identify potential problems, and to refine it. In contrast to the requirements analysis and conceptual design steps, which are essentially subjective, schema refinement can be guided by some elegant and powerful theory. This involves normalizing relations and restructuring them to ensure some desirable properties.

(5) Physical Database Design: In this step we must consider typical expected workloads that our database must support and further refine the database design to ensure that it meets desired performance criteria. This step may simply involve building indexes on some tables and clustering some tables, or it may involve a substantial redesign of parts of the database schema obtained from the earlier design steps.

(6) Security Design: In this step, we identify different user groups and different **roles** played by various users (e.g., the development team for a product, the customer support representatives, and the product manager). For each role and user group, we must identify the parts of the database that they must be able to access and the parts of the database that they should *not* be allowed to access and take steps to ensure that they can access only the necessary parts.

3.2 Entity Relationship Diagram (ERD)

The ER diagram is a semantic data modeling tool that is used to accomplish the goal of abstractly describing or portraying data. Abstractly described data is called a ***conceptual model***. Conceptual model will lead us to a "schema." A ***schema*** implies a permanent, fixed description of the structure of the data. Therefore, when we agree that we have captured the correct depiction of reality within our conceptual model, our ER diagram, we can call it a schema. An ER diagram models data as ***entities*** and ***relationships***, and entities have ***attributes***.

An ENTITY is a thing about which we store data, for example, a person, a bank account, a building. Entity can be a "thing which can be distinctly identified." So an entity can be a person, place, object, event, or concept about which we wish to store data. The name for an entity must be one that represents a type or class of thing not an instance. The name for an entity must be sufficiently generic but, at the same time, the name for an entity cannot be too generic. The name should also be able to accommodate changes "over time." For example, if we were modeling a business and the business made donuts, we might consider creating an entity called DONUT. But how long will it be before this business evolves into making more generic pastry? If it is anticipated that the business will involve pastry of all kinds rather than just donuts, perhaps it would be better to create an entity called PASTRY — it may be more applicable "over time."

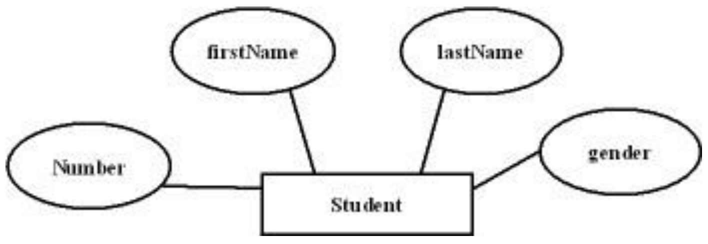
Some examples of entities include:

i. Examples of a person entity would be EMPLOYEE, VET, or STUDENT.
ii. Examples of a place entity would be STATE or COUNTRY.
iii. Examples of an object entity would be BUILDING, AUTO, or PRODUCT.
iv. Example of an event entity would be SALES, RETURNS, or REGISTRATION.
v. Examples of a concept entity would be ACCOUNT or DEPARTMENT

Weak Entity
A weak entity is an entity that depends on the existence of another entity. In more technical terms it can defined as an entity that cannot be identified by its own attributes. It uses a foreign key combined with its attributed to form the primary key. An entity like order item is a good example for this. The order item will be meaningless without an order so it depends on the existence of order.

Attributes
Attributes are the characteristics that describe entities and relationships. For example, a Student entity may be described by attributes including: student number, name, address, date of birth, degree major. An Invoice entity may be described by attributes including: invoice number, invoice date, invoice total. In an ERD using the Chen notation, we illustrate attributes using ovals as shown in figure 1.1 below.

Figure 1.1



Composite attributes, on the other hand, can be divided into subparts (that is, other attributes). For example, an attribute *name* could be structured as a composite attribute consisting of *first-name*, *middle-initial*, and *last-name*. Using composite attributes in a design schema is a good choice if a user will wish to refer to an entire attribute on some occasions and to only a component of the attribute on other occasions. Suppose we were to substitute for the *customer* entity-set attributes *customer-street* and *customer-city* the composite attribute *address* with the Attributes Street, *city*, *state*, and *zip-code*. Composite attributes help us to group together related attributes, making the modeling cleaner. Note also that a composite attribute may appear as a hierarchy. In the composite attribute *address*, its component attribute *street* can be further divided into *street-number*, *street-name*, and *apartment-number*

Single-valued and **multivalued** attributes. The attributes in our examples all have a single value for a particular entity. For instance, the *loan-number* attribute for a specific loan entity refers to only one loan number. Such attributes are said to be **single valued**. There may be instances where an attribute has a set of values for a specific entity. Consider an *employee* entity set with the attribute *phone-number*. An employee may have zero, one, or several phone numbers, and different employees may have different numbers of phones. This type of attribute is said to be **multivalued**. As another example, an at tribute *dependent-name* of the *employee* entity set would be multivalued, since any particular employee may have zero, one, or more dependent(s).

Where appropriate, upper and lower bounds may be placed on the number of values in a multivalued attribute. For example, a bank may limit the number of phone numbers recorded for a single customer to two. Placing bounds in this case expresses that the *phone-number* attribute of the *customer* entity set may have between zero and two values.

Derived attribute. The value for this type of attribute can be derived from the values of other related attributes or entities. For instance, let us say that the *customer* entity set has an attribute *loans-held*, which represents how many loans a customer has from the bank. We can derive the value for this attribute by counting the number of *loan* entities associated with that customer.

As another example, suppose that the *customer* entity set has an attribute *age*, which indicates the customer's age. If the *customer* entity set also has an attribute *date-of-birth*, we can calculate *age* from *date-of-birth* and the current date. Thus, *age* is a derived attribute. In this case, *date-of-birth* may be referred to as a *base* attribute, or a *stored* attribute. The value of a derived attribute is not stored but is computed when required.

Value: Each attribute has a value. An instance of an entity is a mapping from attribute to its value. For example, when we assign value to color, registration ID and make, then we identify a unique car.

Value set (Domain) of attributes: The instance of an entity comes into existence when allowed values are assigned to the attributes of this entity. In order to represent a fact, a value of an attribute comes from a set of values. Such a set whose members can be assigned to an attribute to represent a fact is called attribute *domain* or *value set*.

Attribute hierarchy: An attribute of an object may be made up of several smaller attributes. This means a big attribute may be identified by a set of smaller attributes. In turn, a smaller attribute may be identified by a set of smallest attributes. In this way we could create a hierarchy, which would describe the main attribute. The leaves of this hierarchy cannot be further divided meaningfully, so they are called *atomic* or *simple* attributes.

Relationship

A relationship is a link or association between entities. Relationships are usually denoted by verb phrases, A relationship in an ER diagram is a connection between two or more entities, or between one entity and itself. The latter kind of relationship, between one entity and itself, is known as a *recursive* relationship For example, a *depositor* relationship associates a customer with each account that she has

Degree of a Relationship Type

In reality (a) only two entities are related together or (b) more than two entities are related together. This kind of situation is defined by "degree of a relationship" type. Thus, it defines how many entity types are related with a relationship type.

Binary relation: In this relationship type only two entity types are related with a relationship type. Example: entity type Employee and entity type Department are related with *Works_For* relationship type. Note that only Employee and Department entities are involved in this relation, there is no third entity.

Ternary relation: In this relationship type, three entities (not two and not four) are related with a relationship type.

Constraints on Relationship Types As mentioned earlier, a relationship type can be binary, ternary, etc. We represent these properties in terms of (a) Cardinality ratio and (b) Participation

Cardinality ratio: Number of instances of an entity that can participate in the instance of another entity. For example, **Employee ---- Works_For ---- Department.** If we say that several employees work for a department then this would mean: *One instance of department type is related to several instances of employee type*. So the cardinality ratio of this relationship is 1: N. Cardinality ratios for a binary relationship type: 1:1, 1: N, M: N.

Participation constraint: Specifies that the existence of an entity type depends on the existence of another related entity type. This constraint exists in two ways: (a) **Total participation** and (b) **Partial participation**.

Total participation: This participation exists when all instances of an entity type are related to one instance of an entity type. For example, every employee *works_for* a department. Thus, an employee exists only if a department exists. So the participation of an employee is total in *works_for* relationship. Review: Give another example of total participation between entity types "Project" and "Department".

Partial participation: This participation exists when one of the instances of an entity type is related to an instance of another entity type. For example, one of the employees (Manager) *manages* a department. It is not true that every employee manages a department. This implies that the existence of an employee type does not necessarily mean the existence of *manages* relationship for the employee. So the participation of an employee in manages is partial. Together **Cardinality ratio** and **Participation constraints** are termed as **Structural Constraints**.

The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.

In general, the overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram*, which is built up from the following components:

- **Rectangles**, which represent entity sets
- **Ellipses**, which represent attributes
- **Diamonds**, which represent relationship sets
- **Lines**, which link attributes to entity sets and entity sets to relationship sets
- **Double ellipses**, which represent multivalued attributes
- **Dashed ellipses**, which denote derived attributes

- **Double lines**, which indicate total participation of an entity in a relationship set
- **Double rectangles**, which represent weak entity sets
- Each component is labeled with the entity or relationship that it represents.

Some of the symbols are shown in the figure 1.2

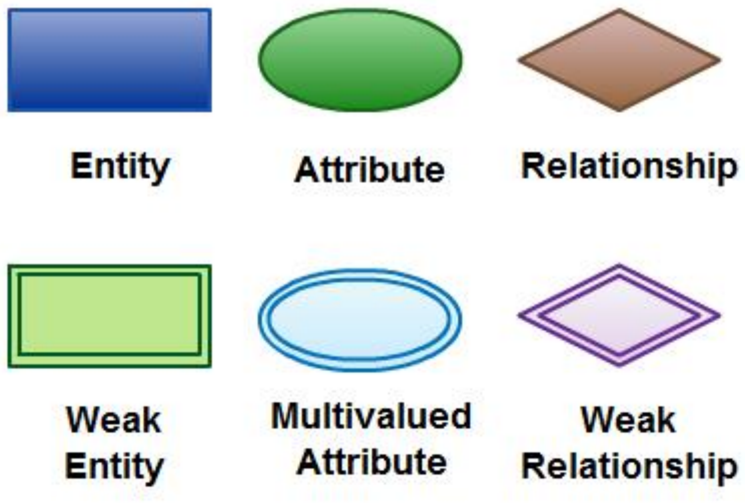


Figure 1.2

In addition to entities and relationships, the E-R model represents certain constraints to which the contents of a database must conform. One important constraint is **mapping cardinalities**, which express the number of entities to which another entity can be associated via a relationship set. For example, if each account must belong to only one customer, the E-R model can express that constraint.

Keys

The sense of a database is to store data for retrieval. An attribute that may be used to find a particular entity occurrence is called a ***key***. As we model our database with the ER models, we may find that some attributes naturally seem to be keys. If an attribute can be thought of as a unique identifier for an entity, it is called a ***candidate key***. When a candidate key is chosen to be ***the*** unique identifier, it becomes the ***primary key*** for the entity. Figure 1.3 shows an example of ER model where primary keys are underlined.

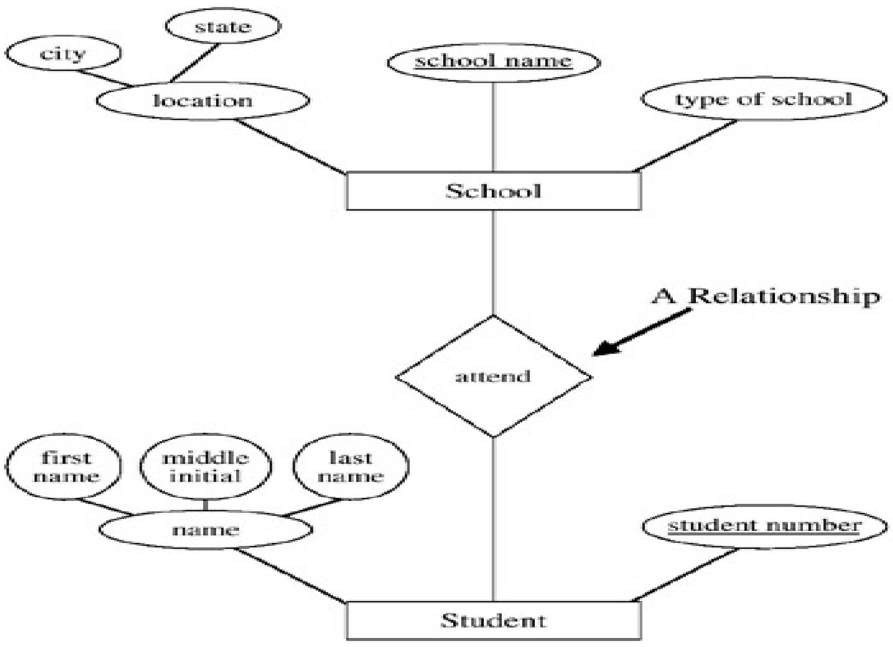


Figure 1.3 ER Model with primary keys underlined

ER Design Methodology

Step 1: Select one, primary entity from the database requirements description and show attributes to be recorded for that entity. Label keys if appropriate and show some sample data.

Step 2: Use structured English for entities, attributes, and keys to describe the database that has been elicited.

Step 3: Examine attributes in the primary entity (possibly with user assistance) to find out if information about one of the entities is to be recorded.

Step 3a: If information about an attribute is needed, then make the attribute an entity, and then

Step 3b: Define the relationship back to the original entity.

Step 4: Show some sample data.

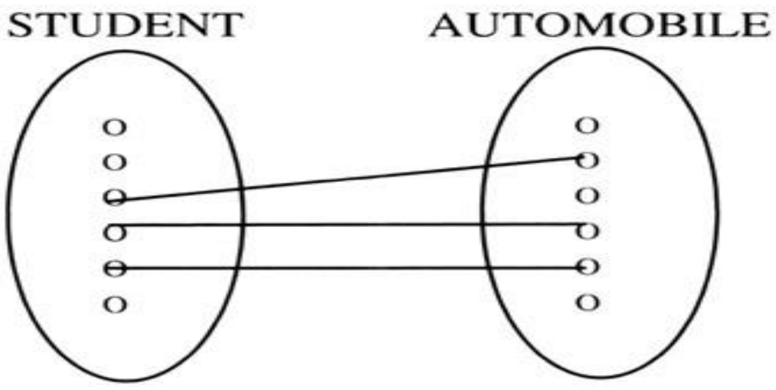
Cardinality Ratio of a Relationship

Cardinality is a rough measure of the number of entities (one or more) that will be related to another entity (or entities). For example, there are four ways in which the entities AUTOMOBILE and STUDENT can be "numerically involved" in a relationship: one-to-one (1:1), many-to-one (M:1), one-to-many(1:M), and many-to-many (M:N).

One-to-One (1:1)

In this type of relationship, one entity is associated with one other entity, and vice versa. Take, for example, if in our drive relationship (shown in Figure below), we stated that one automobile is driven by one student and one student drives one automobile, then the student/automobile relationship would be one-to-one, symbolically:

STUDENT:AUTOMOBILE :: 1:1



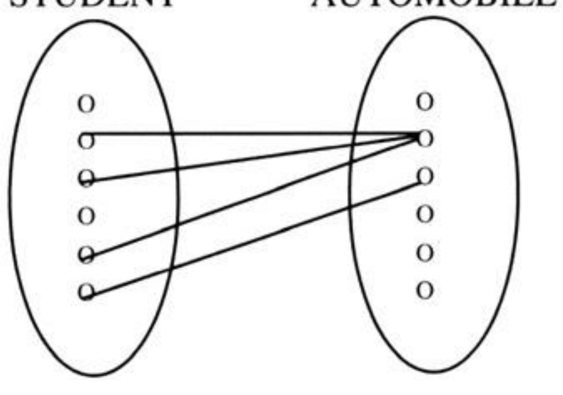
Many-to-One (M:1)

If the SA (STUDENT:AUTOMOBILE) relationship were many-to-one, then we would say that many students are associated with one automobile and one automobile is associated with many students; that is:

STUDENT:AUTOMOBILE::M:1

We have intentionally used the verb phrase "is associated with" in place of drive because the statement "many students drive one automobile" can be taken in a variety of ways. Also, using a specific verb for a relationship is not always best when the diagram is first drawn, unless the analyst is absolutely sure that the verb correctly describes the user's intention. We could have also used the verb phrase "is related to" instead of "is associated with" if we wanted to be uncommitted about the exact verb to use.

We will tighten the language used to describe relationships presently, but what does an STUDENT:AUTOMOBILE::M:1 relationship imply? It would represent a situation where perhaps a family owned one car and that carwash driven by multiple people in the family.

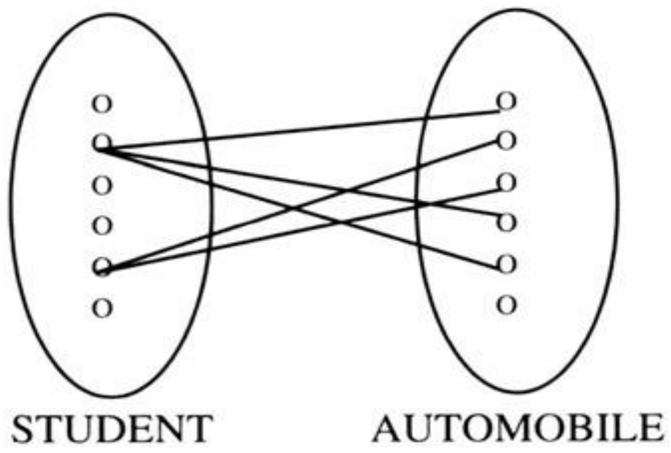


One-to-Many (1:M)

The sense of a one-to-many SA (STUDENT:AUTOMOBILE) relationship(shown in Figure 3.6) would be that a student is associated with many automobiles and an automobile is associated with one student. Clearly, if we define a relationship as 1:M (or M:1), then we need to be very clear about

which entity is 1 and which is M. Here:

STUDENT:AUTOMOBILE::1:M



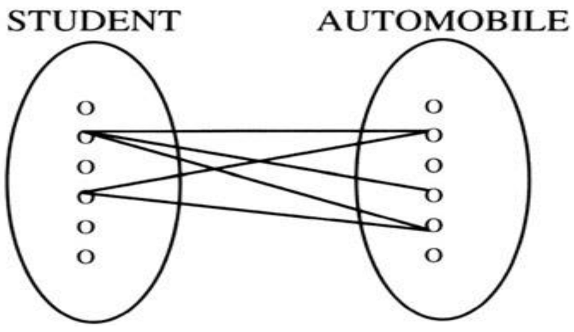
Many-to-Many (M:N)

In many-to-many relationships, many occurrences of one entity are associated with many of the other. Many-to-many is depicted as M:N, as in M of one thing related to N of another thing. Older database texts called this an M:M relationship, but newer books use M:N to indicate that the number of things related is not presumed to be equal (the values of M and N are likely to be different).

If our SA relationship were many-to-many, a student would be associated with many automobiles and an automobile with many students:

STUDENT: AUTOMOBILE:: M:N

In this case (if we assumed SA = drive, as shown in Figure 3.6), multiple students can drive multiple cars (hopefully not all at the same time) and multiple cars can be driven by multiple students. Picture, for example, a family that has multiple cars and any one family member can drive any of the cars and any car can be driven by any family member



Self Assessment Exercise(s)

1. Explain the difference between a weak entity set and a strong entity set.
2. What are the constraints used in E-R model?
3. An association between two or more entities is known as
4. A rough measure of the number of entities (one or more) that will be related to another entity (or entities) is.

Self Assessment Answer(s)

4.0 Conclusion

In this unit we have considered the steps involve in database design and a comprehensive review of ER model.

5.0 Summary

You have learnt that:

1. Database design has six steps: requirements analysis, conceptual database design, logical database design, schema refinement, physical database design, and security design. Conceptual design should produce a high-level description of the data, and the entity-relationship (ER) data model provides a graphical approach to this design
2. In the ER model, a real-world object is represented as an *entity*.
3. An *entity set* is a collection of structurally identical entities. Entities are described using *attributes*.
4. Each entity set has a distinguished set of attributes called a *key* that can be used to uniquely identify each entity.
5. A *relationship* is an association between two or more entities. A *relationship set* is a collection of relationships that relate entities from the same entity sets. A relationship can also have *descriptive attributes*.
6. A *key constraint* between an entity set S and a relationship set restricts instances of the relationship set by requiring that each entity of S participate in at most one relationship. A *participation constraint* between an entity set S and a relationship set restricts instances of the relationship set by requiring that each entity of S participate in at least one relationship.
7. The identity and existence of a *weak entity* depends on the identity and existence of another (*owner*) entity. *Class hierarchies* organize structurally similar entities through inheritance into sub- and super classes.
8. *Aggregation* conceptually transforms a relationship set into an entity set such that the resulting construct can be related to other entity sets.

9. Development of an ER diagram involves important modeling decisions. A thorough understanding of the problem being modeled is necessary to decide whether to use an attribute or an entity set, an entity or a relationship set, a binary or ternary relationship, or aggregation.

6.0 Tutor-Marked Assignment

1. A company database needs to store information about employees (identified by *ssn*, with *salary* and *phone* as attributes), departments (identified by *dno*,with *dname* and *budget* as attributes), and children of employees (with *name* and *age* as attributes). Employees *work* in departments; each department is *managed by* an employee; a child must be identified uniquely by *name* when the parent (who is an employee; assume that only one parent works for the company) is known. We are not interested in information about a child once the parent leaves the company.

Draw an ER diagram that captures this information.

2. Define the following: Composite attribute, Multivalued attribute, Derived attribute

7.0 References/ Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://creately.com/blog/diagrams/er-diagrams-tutorial/

http://www.edrawsoft.com/chen-erd.php

http://ion.uwinnipeg.ca/~rmcfadye/2914/hypergraph/relationshipType.html

http://www.nou.edu.ng/noun/NOUN_OCL/pdf/pdf2/CIT%20744.pdf

http://bitcse.selfip.com/LECTURE%20NOTES/DBMS_SN/1.Overview.pdf

http://my.safaribooksonline.com/book/databases/9788131731925/database-system/ch01lev1sec8

Unit 2

Relational Database

Content

1.0 Introduction

Relational database can be defined as a collection of data items organized in a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database In addition to being relatively easy to create and access, a relational database has the important advantage of being easy to extend. After the original database creation, a new data category can be added without requiring that all existing applications be modified.

2.0 Learning Outcomes

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

i. State the meaning of relational database
ii. Describe table, relations, tuple and instance
iii. Carry out mapping from ER to Relational model

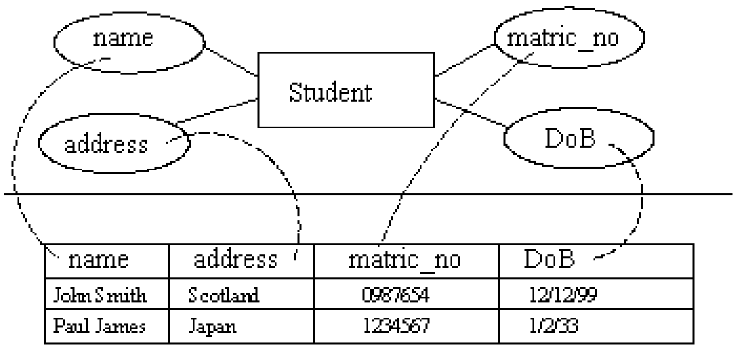
3.0 Learning Contents

3.1 Relational Databases

A relational database consists of a collection of **tables**, each of which is assigned a unique name. Basically, a relational database is a database of two-dimensional tables called "relations." The tables are composed of rows and columns. The rows are often called ***tuples*** and the columns, ***attributes***. In relational databases, all attributes (table columns) must be atomic and keys must not be null. In addition, in relational databases, the actual physical location of the data on a disk is not usuallynecessary to know.

An **instance** of a relation is a set of **tuples**, also called **records**, in which each tuple has the same number of fields as the relation schema. A relation instance can be thought of as a *table* in which each tuple is a *row*, and all rows have the same number of fields. (The term *relation instance* is often abbreviated to just *relation*, when there is no confusion with other aspects of a relation such as its schema.).

Figure 2.1 below show an ER and the corresponding database table with some tuple.



name	address	matric_no	DoB
John Smith	Scotland	0987654	12/12/99
Paul James	Japan	1234567	1/2/33

Figure 2.1 a sample table

Mapping ER diagram into Relational database

The process of converting an ER diagram into a database is called ***Mapping***

We will consider mapping rules to map ER diagrams into relational databases.

We start with a rule to map strong entities:

M1 — for strong entities: develop a new table (relation) for each strong entity and make the indicated key of the strong entity the primary key of the table. If more than one candidate key is indicated on the ER diagram, choose one for the primary key.

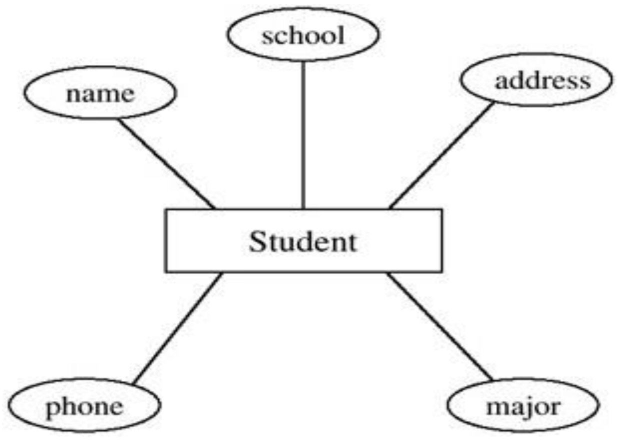
Next, we must map the attributes into the strong entity. Mapping rules are different for atomic attributes, composite attributes, and multi-valued attributes. First, we present the mapping rule for mapping atomic attributes:

M1a — mapping atomic attributes from an entity — for entities with atomic attributes: map entities to a table by forming columns for the atomic attributes

Example:

A relational database realization of the ER diagram in Figure 2.2 below with some data would look like this:

Figure 2.2: ER model



STUDENT(name, phone, school,address,major)

STUDENT

Name	**Phone**	**school**	**address**	**major**
Jones	08032521233	St James	Akure	Chemistry
Smith	08756979797.	St Paul	Ondo	Maths
Adams	0876345562 1	St Theresa	Minna	Agriculture
Sumon	08434575657	OOU	Kaduna	Comp Sc
Mala	08063867664	FUT	Owerri	Agriculture

Table 2.1

The entity name, STUDENT, would be the name of the relation (table). The attributes in the ER diagram become the column headings. The actual table with data, a realization of a relation, is provided as an example of the type of data you might expect from such a relation. The ordering of the columns is irrelevant to relational database as long as once the ordering is chosen.

What about the composite and multi-valued attributes? As mentioned above, it is an axiom of relational databases that all columns be atomic. If we have a non-atomic attribute in our diagram, we have to make it atomic for mapping to the relational database. For composite attributes, we achieve atomicity by recording only the component parts of the attribute.

Our next mapping rule concerns composite attributes:

M1b – for entities with composite attributes: map entities to a table by forming columns from the elementary (atomic) parts of the composite attributes.

M1c — for multi-valued attributes: form a separate table for the multi-valued attribute. Record a row for each value of the multivalued attribute, together with the key from the original table. The key of the new table will be the concatenation of the multivalued attribute plus the key of

the owner entity. Remove the multi-valued attribute from the original table**.**

3.2 Integrity Constraints over Relations

A database is only as good as the information stored in it, and a DBMS must therefore help prevent the entry of incorrect information. An **integrity constraint (IC)** is a condition that is specified on a database schema and restricts the data that can be stored in an instance of the database. If a database instance satisfies all the integrity constraints specified on the database schema, it is a **legal** instance. A DBMS **enforces** integrity constraints, in that it permits only legal instances to be stored in the database.

Integrity constraints are specified and enforced at different times:

1. When the DBA or end user defines a database schema, he or she specifies the ICs that must hold on any instance of this database.
2. When a database application is run, the DBMS checks for violations and disallows changes to the data that violate the specified ICs. (In some situations, rather than disallow the change, the DBMS might instead make some compensating changes to the data to ensure that the database instance satisfies all ICs. In any case, changes to the database are not allowed to create an instance that violates any IC.)

Many kinds of integrity constraints can be specified in the relational model. We have already seen one example of an integrity constraint in the *domain constraints* associated with a relation schema (Section 3.1). In general, other kinds of constraints can be specified as well; for example, no two students have the same *sid* value. In this section we discuss the integrity constraints, other than domain constraints, that a DBA or user can specify in the relational model.

Key Constraints

Consider the Students relation and the constraint that no two students have the same student id. This IC is an example of a key constraint. A **key constraint** is a statement that a certain *minimal* subset of the fields of a relation is a unique identifier for a tuple.

A set of fields that uniquely identifies a tuple according to a key constraint is called a **candidate key** for the relation; we often abbreviate this to just *key*. In the case of the Students relation, the (set of fields containing just the) *sid fi*eld is a candidate key.

Let us take a closer look at the above definition of a (candidate) key. There are two parts to the definition:

1. Two distinct tuples in a legal instance (an instance that satisfies all ICs, including the key constraint) cannot have identical values in all the fields of a key.
2. No subset of the set of fields in a key is a unique identifier for a tuple.

The first part of the definition means that in *any* legal instance, the values in the key fields uniquely identify a tuple in the instance. When specifying a key constraint, the DBA or user must be sure that this constraint will not prevent them from storing a correct set of tuples. (A similar comment applies to the specification of other kinds of ICs as well.) The notion of `correctness' here depends upon the nature of the data being stored. For example, several students may have the same name, although each student has a unique student id. If the *name fi*eld is declared to be a key, the DBMS will not allow the Students relation to contain two tuples describing different students with the same name!

The second part of the definition means, for example, that the set of fields *fsid, nameg* is not a key for Students, because this set properly contains the key *fsidg*. The set*fsid, nameg* is an example of a **super key**, which is a set of fields that contains a key.

Look again at the instance of the Students relation in Table 2.1. Observe that two different rows always have different *sid* values; *sid* is a key and uniquely identifies a tuple. However, this does not hold for nonkey fields. For example, the relation contains two rows with *Smith* in the *name fi*eld.

Note that every relation is guaranteed to have a key. Since a relation is a set of tuples, the set of *all fi*elds is always a super key. If other constraints hold, some subset of the fields may form a key, but if not, the set of all fields is a key.

A relation may have several candidate keys. For example, the *login* and *age fi*elds of the Students relation may, taken together, also identify students uniquely. That is, *login, ageg* is also a key. It may seem that *login* is a key, since no two rows in the example instance have the same *login* value. However, the key must identify tuples uniquely in all possible legal instances of the relation. By stating that *flogin, ageg* is a key, the user is declaring that two students may have the same login or age, but not both.

Out of all the available candidate keys, a database designer can identify a **primary** key. Intuitively, a tuple can be referred to from elsewhere in the database by storing the values of its primary key fields. For example, we can refer to a Students tuple by storing its *sid* value. As a consequence of referring to student tuples in this manner, tuples are frequently accessed by specifying their *sid* value. In principle, we can use any key, not just the primary key, to refer to a tuple. However, using the primary key is preferable because it is what the DBMS expects this is the significance of designating a particular candidate key as a primary key and optimizes for. For example, the DBMS may create an index with the primary key fields as the search key, to make the retrieval of a tuple given its primary key value efficient. The idea of referring to a tuple is developed further in the next section.

Foreign Key Constraints

Sometimes the information stored in a relation is linked to the information stored in another relation. If one of the relations is modified, the other must be checked, and perhaps modified, to keep the data consistent. An IC involving both relations must be specified if a DBMS is to make such checks. The most common IC involving two relations is a *foreign key* constraint.

Suppose that in addition to Students, we have a second relation:

Enrolled(*sid:* string, *cid:* string, *grade:* string)

To ensure that only bona fide students can enroll in courses, any value that appears in the *sid fi*eld of an instance of the Enrolled relation should also appear in the *sid* field of some tuple in the Students relation. The *sid* field of Enrolled is called a **foreign key** and **refers** to Students. The foreign key in the referencing relation (Enrolled, in our example) must match the primary key of the referenced relation (Students), i.e., it must have the same number of columns and compatible data types, although the column names can be different.

Example of a relational database

Assume that we wish to maintain a database that contains the flight information of an airline during one particular year. This database maintains information about flights, pilots, and assignments of pilots to flights. Each flight has a unique flight number, a departure city, a destination city, a departure time, and an arrival time. Each pilot has a unique company ID,a name, and an experience level. Pilots are assigned to y certain flights on particular days of the year.

1. Using this description of the database, identify the data objects and a relationship that need to be maintained in the database.

Data objects: flights and pilots

Relationship: assigned-to, i.e, pilots are assigned to flights.

2. For each data object specify its relevant attributes and identify its primary key.

Flight: flight number, departure city, destination city, departure time, and arrival time. The primary key is the flight number attribute.

Pilot: company ID, name, and experience level. The primary key is the company attribute.

3. For the relationship, determine its attribute(s) and identify its foreign keys.

Assigned-to: flight number, company ID, and date. The key is ofAssigned-to is (flight number, company ID)

Assigned-to as two foreign keys. The Assigned-to.(flight number) attribute is a foreign key referencing the Flight.(flight number) key attribute. And, the Assigned-to.(company ID) attribute is a foreign key referencing the Pilot.(company ID) key attribute.

4. Use relational tables to represent the data objects and relationships with their attributes and constraints.

Flight

flight number	dept. city	dest. City	dept. time	arrival time

Pilot

company ID	Name	experience level

Assigned-to

company ID	Flight number	date

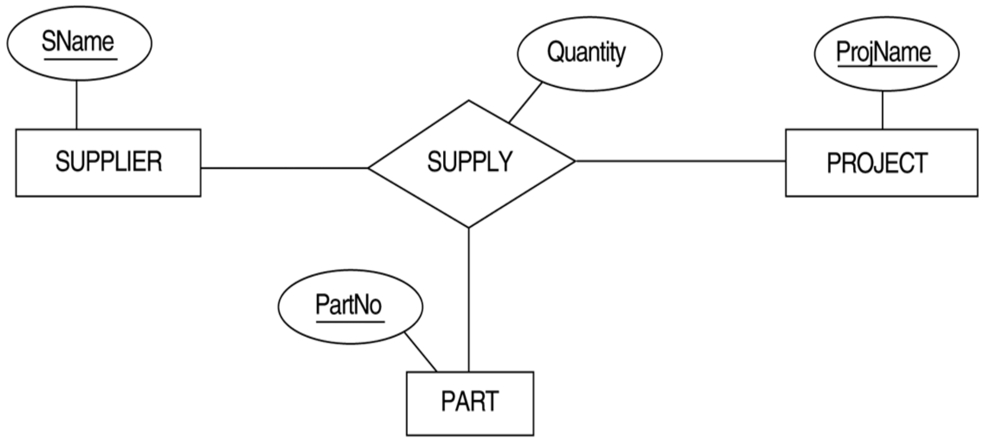
Primary keys are underlined ones. Foreign keys are underlined twice.

The foreign key company ID in the Assigned-to table references the primary key company id in the Pilot table.

The foreign key flight number in the Assigned-to table references

The primary key flight number in the Flight table.

Example 2: Given the ER model below draw the corresponding relation schema



The relational model is shown below

SUPPLIER(SName, ...)

PROJECT(ProjName, ...)

PART(PartNo, ...)

SUPPLY(SName, ProjName, PartNo, Quantity)

Self Assessment Exercise(s)

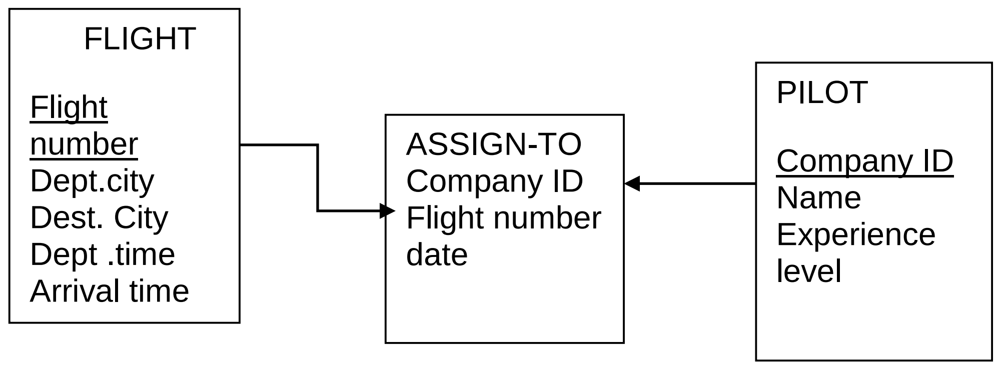
1. The rows in a relational database are often called ------While the column is ***tuples*** and the columns, ***attributes***

Self Assessment Answer(s)

Schema Diagram

A database schema, along with primary key and foreign key dependencies, can be depicted pictorially by **schema diagram**s. Each relation appears as a box, with the attributes listed inside it and the relation name above it. If there are primary key attributes, a horizontal line crosses the box, with the primary key attributes listed above the line. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation. Do not confuse a schema diagram with an E-R diagram. In particular, E-R diagrams do not show foreign key attributes explicitly, whereas schema diagrams show them explicitly.

Many database systems provide design tools with a graphical user interface for creating schema diagrams



The schema diagram for example 1 above is shown in figure 2.3

Figure 2.3: Schema diagram

Self Assessment Exercise(s)

1. What is the basic difference between E-R diagram and Schema diagram?

Self Assessment Answer(s)

4.0. Conclusion

In this unit you have the concept of relational database, keys, mapping of ER to relational model and schema diagram.

5.0. Summary

You have learnt:

1. The main element of the relational model is a *relation*.
2. A *relation schema* describes the structure of a relation by specifying the relation name and the names of each field. In addition, the relation schema includes *domain constraints*, which are type restrictions on the fields of the relation.
3. The number of fields is called the *degree* of the relation.
4. The *relation instance* is an actual table that contains a set of *tuples* that adhere to the relation schema.
5. *Integrity constraints* are conditions on a database schema that every legal database instance has to satisfy.
6. Besides domain constraints, other important types of ICs are *key constraints* (a minimal set of fields that uniquely identify a tuple) and *foreign key constraints* (fields in one relation that refer to fields in another relation)

6.0 Tutor Marked Assignment

1. You want to create a database about businesses. Each business will have a name, an address, the business phone number, the owner's phone number, and the first names of the employees who work at the business. Draw the ER diagram using the Chen-like model, state any assumptions you made when drawing the diagrams. Map your diagrams to a relational database.
   a. Which attributes would you consider composite attributes in this database?
   b. Which attributes would you consider multi-valued attributes in this database?
   c. Could there be any derived attributes? What would be good keys?
2. You want to create a database about the books on your shelf. Each book has authors (assume last name only is needed), title, publisher, courses used in (course number only). Draw the ER diagram using the Chen-like model. State any assumptions you made when drawing the diagrams.
   a. Which attributes would you consider composite attributes in this database?
   b. Which attributes would you consider multi-valued attributes in this database?
3. Could there be any derived attributes? What would be good keys? Map your diagram to a relational database and draw the schema diagram

7.0 References/ Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://searchsqlserver.techtarget.com/definition/relational-database

http://web.eecs.umich.edu/~teorey/lec.notes.pdf

http://coronet.iicm.tugraz.at/Dbase1/scripts/rdbh04.htm

http://creately.com/blog/diagrams/er-diagrams-tutorial/

http://www.edrawsoft.com/chen-erd.php

http://ion.uwinnipeg.ca/~rmcfadye/2914/hypergraph/relationshipType.html

Unit 3

Functional Dependency and Normalisation

Contents

1.0 Introduction

It is necessary to know how to design databases that are normalised - the process of breaking data down into its most basic components. We do this to remove redundancy in data, and for a couple of reasons. Data is more flexible when in a granular format. Instead of using one big table, like with any spreadsheet program, we make distinctions between bits of data and then relate these bits to create meaningful information. This unit takes a look at this concept

2.0. Learning Outcomes

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

i. State the meaning of functional/database dependency
ii. Explain different types of anomaly
iii. Explain the meaning normalization
iv. Describe how to normalize database in 1NF,2NF, 3NF and BCNF

3.0 Learning Contents

3.1 Normalisation

Normalisation is the process for evaluating and correcting table structures to minimize data redundancies thereby helping to eliminate data anomalies. Normalisation works through a series of stages called Normal forms. The first three stages are described as first normal form(1NF), second normal form (2NF) and third normal form (3NF). The concept was introduced by professor Codd in 1970 when he defined the first three normal forms (first, second and third normal forms). Normalization is used to avoid or eliminate the three types of anomalies (insertion, deletion and update anomalies) which a database may suffer from.

Data normalization is a set of rules and techniques concerned with:

i. Identifying relationships among attributes.
ii. Combining attributes to form relations.
iii. Combining relations to form a database.

It follows a set of rules worked out by E FCodd in 1970. A normalized relational database provides several benefits:

i. Elimination of redundant data storage.
ii. Close modeling of real world entities, processes, and their relationships.
iii. Structuring of data so that the model is flexible.

Because the principles of normalization were first written using the same terminology as was used to define the relational data model this led some people to think that normalization is difficult. Nothing could be more untrue. The principles of normalization are simple, common sense ideas that are easy to apply.

Definition of the three types of anomalies:

i. Insertion anomaly means that that some data cannot be inserted in the database.
ii. Update anomaly means we have data redundancy in the database and to make any modification we have to change all copies of the redundant data or else the database will contain incorrect data. To change its description to "New Database Concepts" we have to change it in all places. Indeed one of the purposes of normalization is to eliminate data redundancy in the database.
iii. Deletion anomaly means deleting some data cause other information to be lost

3.2 Database/ Functional Dependencies

A dependency occurs in a database when information stored in the same database table uniquely determines other information stored in the same table. You can also describe this as a relationship where knowing the value of one attributes (or a set of attributes) is enough to tell you the value of another attribute (or set of attributes) in the same table.

A functional dependency occurs when one attribute in a relation uniquely determines another attribute. This can be written A -> B which would be the same as stating "B is functionally dependent upon A."

Saying that there is a dependency between attributes in a table is the same as saying that there is a functional dependency between those attributes. If there is a dependency in a database such that attributes B is dependent upon attribute A, you would write this as “A -> B”.

For example, in a table listing employee characteristics including Social Security Number (SSN) and name, it can be said that name is dependent upon SSN (or SSN -> name) because an employee's name can be uniquely determined from their SSN. However, the reverse statement (name -> SSN) is not true because more than one employee can have the same name but different SSNs.

Trivial Functional Dependencies

A **trivial functional dependency** occurs when you describe a functional dependency of an attribute on a collection of attributes that includes the original attribute. For example, “{A, B} -> B” is a trivial functional dependency, as is “{name, SSN} -> SSN”. This type of functional dependency is called trivial because it can be derived from common sense. It is obvious that if you already know the value of B, then the value of B can be uniquely determined by that knowledge.

Full Functional Dependencies

A **full functional dependency** occurs when you already meet the requirements for a functional dependency and the set of attributes on the left side of the functional dependency statement cannot be reduced any farther. For example, “{SSN, age} -> name” is a functional dependency, but it is not a full functional dependency because you can remove age from the left side of the statement without impacting the dependency relationship.

Transitive Dependencies

Transitive dependencies occur when there is an indirect relationship that causes a functional dependency. For example, "A -> C" is a transitive dependency when it is true only because both "A -> B" and "B -> C" are true.

Multivalued Dependencies

Multivalued dependencies occur when the presence of one or more rows in a table implies the presence of one or more other rows in that same table. For example, imagine a car company that manufactures many models of car, but always makes both red and blue colors of each model. If you have a table that contains the model name, color and year of each car the company manufactures, there is a multivalued dependency in that table. If there is a row for a certain model name and year in blue, there must also be a similar row corresponding to the red version of that same car.

Example

In a table listing employee characteristics including Employee Number (EmpN) and name, it can be said that name is functionally dependent upon EmpN (or EN -> name) because an employee's name can be uniquely determined from their EmpN. However, the reverse statement (name -> SSN) is not true because more than one employee can have the same name but different EmpNs.

The main characteristics of functional dependencies that we use in normalization:

i. have a one-to-one relationship between attribute(s) on the left and right-hand side of a dependency;
ii. hold for all time;
iii. are nontrivial.

3.3 First Normal Form (1NF)

The first normal form (1NF) requires that data in tables be two-dimensional — that there be no repeating groups in the rows. In other words, none of the attributes of the relation is a relation. Notice that relation means 2-dimentionaltable. An example of a table not in 1NF is where there is an employee "record" such as:

Employee (name, address, {dependent name})

where {dependent name} infers that the attribute is repeated. Sample data for this record might be:

Smith, 123 4th St., {John, Mary, Paul, Sally}

Jones, 4 Moose Lane. {Edgar, Frank, Bob}

Adams, 88 Tiger Circle. {Kaitlyn, Alicia, Allison}

The problem with putting data in tables with repeating groups is that the table cannot be easily indexed or arranged so that the information in the repeating group can be found without searching each record individually.

Relational people usually call a repeating group "nonatomic" (it has more than one value and can be broken apart).

So, the term first normal form (1NF) describes the tabular format in which:

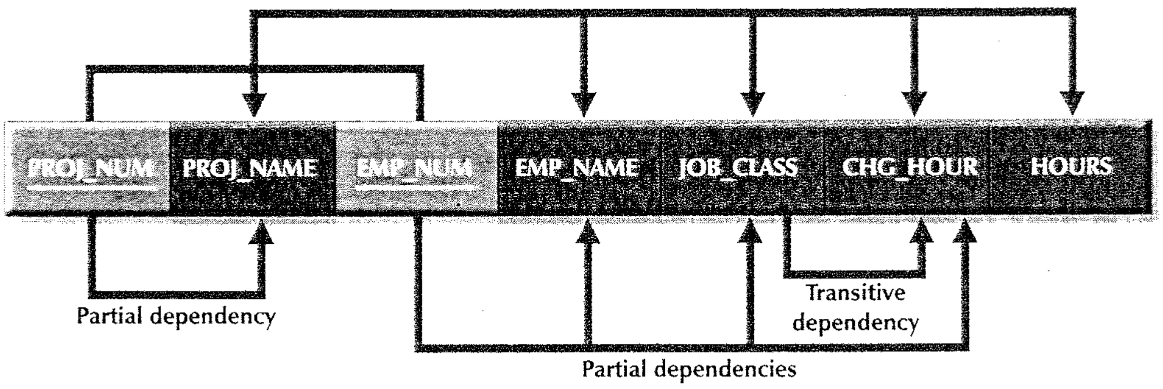
All the key attributes are defined

There are no repeating groups in the table i.e. each row/column intersection can contain one and only one value

All attributes are dependent on the primary key.

Figure 3.1 shows a dependency diagram

Figure 3.1:A dependency diagram: 1NF



3.4 Second Normal Form (2NF)

The second normal form (2NF) requires that data in tables depends on the whole key of the table. Partial dependencies are not allowed. A table is in second normal form if It is in 1NF and it includes no partial dependencies that is no attribute is dependent on only a portion of the primary key

An example:

Employee (name, job, salary, address)

where it takes a name + job combination (a concatenated key) to identify a salary, but address depends only on name. Some sample data are given in table 3.1:

Name	**Job**	**Salary**	**Address**
Smith	Welder	14.75	123 4th St
Smith	Programmer	24.50	123 4th St
Smith	Waiter	7.50	123 4th St
Jones	Programmer	26.50	4 Moose Lane
Jones	Bricklayer	34.50	4 Moose Lane
Adams	Analyst	28.50	88 Tiger Circle

Table 3.1

The problem here is obvious. The address would be repeated for each occurrence of a name. This repeating is called *redundancy* and leads to *anomalies.* An anomaly means that there is a restriction on doing something due to the arrangement of the data.

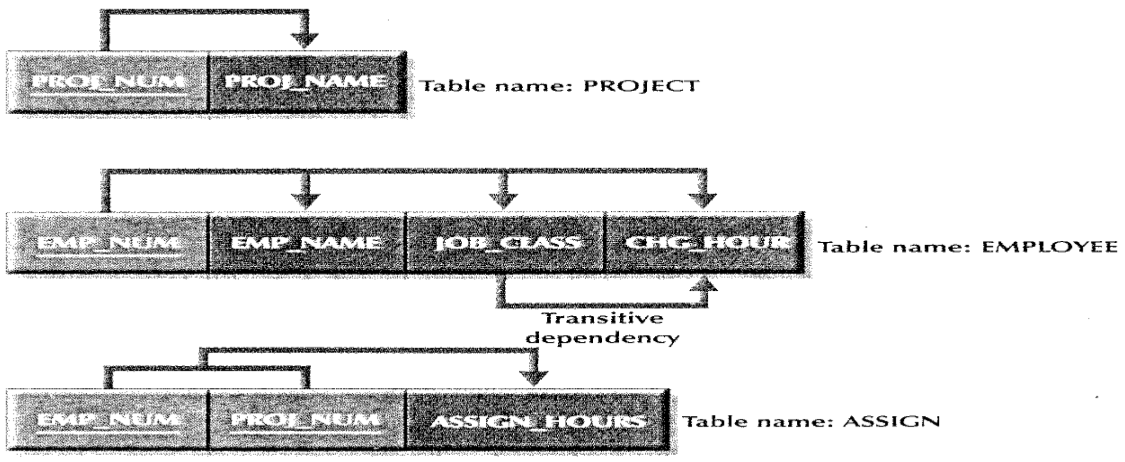
There are insertion anomalies, deletion anomalies, and update anomalies. The key of this table is Name + Job — this is clear because neither one is unique nor it really takes both name and job to identify a salary. However, address depends only on the name, not the job; this is an example of a partial dependency.

Address depends on only part of the key.

An example of an insertion anomaly would be where one would want to insert a person into the table above, but the person to be inserted is not yet assigned a job. This cannot be done because a value would have to be known for the job attribute. Null values cannot be valid values for keys in relational databases (this is known as the entity-integrity constraint).

An update anomaly would be where one of the employees changed his or her address. Three rows would have to be changed to accommodate this one change of address. An example of a delete anomaly would be that Adams quits, so Adams is lost, but then the information that the analyst is being paid N28.50 is also lost. Therefore, more related information than was previously anticipated is lost. Figure 3.2 shows the second normal form.

Figure 3.2 Second Normal form



3.5 Third Normal Form (3NF)

The third normal form (3NF) requires that the data in tables depends on the primary key of the table. A table in third normal form (3NF) if it is in 2NF and it contains no transitive dependencies. i.e Already meet the requirements of both 1NF and 2NF Remove columns that are not fully dependent upon the primary key.

A classic example of non-3NF is:

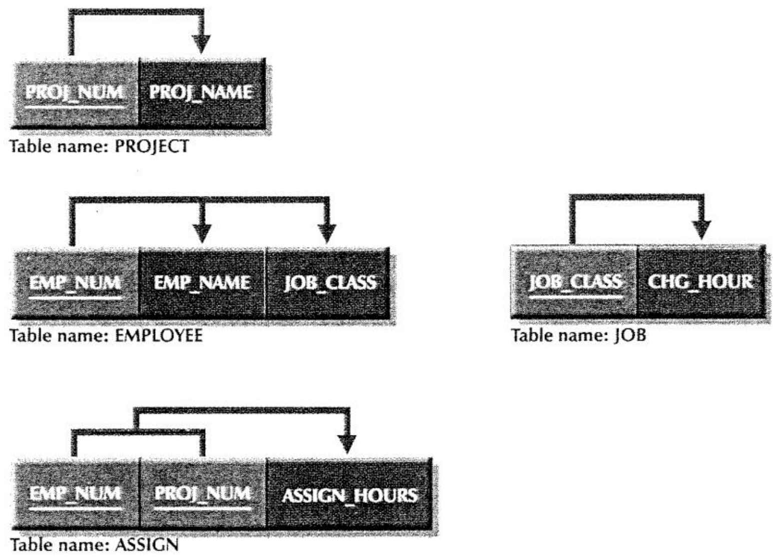
Employee (name, address, project#, project-location)

Suppose that project-location means the location from which a project is controlled, and is defined by the project#. Some sample data will show the problem with this table:

Name	**Address**	**Project#**	**Project-location**
Smith	123 4th St	101	Memphis
Smith	123 4th St	102	Mobile
Jones	4 Moose Lane	101	Memphis

Note the redundancy in this table. Project 101 is located in Memphis; but every time a person is recorded as working on project 101, the fact that they work on a project that is controlled from Memphis is recorded again. The same anomalies — insert anomaly, update anomaly, and delete anomaly —are also present in this table.

To clear the database of anomalies and redundancies, databases must be normalized. The normalization process involves splitting the table into two or more tables (a decomposition). After tables are split apart (a process called decomposition), they can be reunited with an operation called a "join." There are three decompositions that would alleviate the normalization problems in our examples, as discussed below.



Examples of 1NF, 2NF, and 3NF

Example of Non-1NF to 1NF

Here, the repeating group is moved to a new table with the key of the table from which it came.

Non-1NF:

Smith, 123 4th St., {John, Mary, Paul, Sally}

Jones, 4 Moose Lane., {Edgar, Frank, Bob}

Adams, 88 Tiger Circle., {Kaitlyn, Alicia, Allison}

is decomposed into 1NF tables with no repeating groups:

Employee table

Name	Address
John	151 5th St
Jones	4 Moose Lane
Adams	88 Tiger Circle

Dependent Table

DependentName	**EmployeeName**
John	Smith
Mary	Smith
Paul	Smith
Sally	Smith
Edgar	Jones
Frank	Jones
Kaitlyn	Adams
Alicia	Adams
Allison	Adams

In the EMPLOYEE table, Name is defined as a key — it uniquely identifies the rows. In the DEPENDENT table, the key is a combination (concatenation) of DependentName and EmployeeName. Neither the DependentName nor the EmployeeName is unique in the DEPENDENT table, and therefore both attributes are required to uniquely identify a row in the table. The EmployeeName in the DEPENDENT table is called a foreign key because it references a primary key, Name in another table, the EMPLOYEE table. Note that the original table could be reconstructed by combining these two tables by recording all the rows in the EMPLOYEE table and combining them with the corresponding rows in the

EMPLOYEE table where the names were equal (an equi-join operation). Note that in the derived tables, there are no anomalies or unnecessary redundancies.

Example of Non-2NF to 2NF

Here, partial dependency is removed to a new table.

Non-2NF:

Name	Job	Salary	Address
Smith	Welder	14.75	123 4th St
Smith	Programmer	24.50	123 4th St
Smith	Waiter	7.50	123 4th St
Jones Lane	Programmer	26.50	4 Moose
Jones Lane	Bricklayer	34.50	4 Moose
Adams Circle	Analyst	28.50	88 Tiger

is decomposed into 2NF:

Name + Job table

NAME AND JOB

Name	Job	Salary
Smith	Welder	14.75
Smith	Programmer	24.50
Smith	Waiter	7.50
Jones	Programmer	26.50
Jones	Bricklayer	34.50
Adams	Analyst	28.50

Name and Address (Employee info) table:

Name	Address
Smith	123 4th St
Jones	4 Moose Lane
Adams	88 Tiger Circle

Again, note the removal of unnecessary redundancy and the amelioration removal of possible anomalies.

Example of Non-3NF to 3NF

Here, transitive dependency is removed to a new table.

Non-3NF:

Name	Address	Project#	Project-location
Smith	123 4th St	101	Memphis
Smith	123 4th St	102	Mobile
Jones	4 Moose Lane	101	Memphis

is decomposed into 3NF:

EMPLOYEE table:

EMPLOYEE

Name	Address	Project#
Smith	123 4th St	101
Smith	123 4th St	102
Jones	4 moose Lane	101

PROJECT table:

Project#	Project-location
101	Memphis
102	Mobile
101	Memphis

Again, observe the removal of the transitive dependency and the anomaly problem. There are more esoteric normal forms, but most databases will be well constructed if they are normalized to the 3NF. The intent here is to show the general process and merits of normalization.

3.6 Boyce-Codd Normal Form (BCFN)

A relation R is in Boyce-Codd normal form (BCNF) if and only if every determinant is a candidate key. The definition of BCNF addresses certain (rather unlikely) situations which 3NF does not handle. The characteristics of a relation which distinguish 3NF from BCNF are given below. Since it is so unlikely that a relation would have these characteristics, in practical real-life design it is usually the case that relations in 3NF are also in BCNF. Thus, many authors make a "fuzzy" distinction between 3NF and BCNF when it comes to giving advice on "how far" to normalize a design. Since relations in 3NF but not in BCNF are slightly unusual, it is a bit more difficult to come up with meaningful examples. To be precise, the definition of 3NF does not deal with a relation that:

i. has multiple candidate keys, where
ii. those candidate keys are composite, and
iii. the candidate keys overlap (i.e., have at least one common attribute)

Example:

Grade_report(StudNo,StudName,(Major,Adviser,

(CourseNo,Ctitle,InstrucName,InstructLocn,Grade)))

- Functional dependencies

StudNo -> StudName

CourseNo -> Ctitle,InstrucName

InstrucName -> InstrucLocn

StudNo,CourseNo,Major -> Grade

StudNo,Major -> Advisor

Advisor -> Major

- Unnormalised

Grade_report(StudNo,StudName,(Major,Advisor,

(CourseNo,Ctitle,InstrucName,InstructLocn,Grade)))

- 1NF Remove repeating groups

Student(<u>StudNo</u>,StudName)

StudMajor(<u>StudNo,Major</u>,Advisor)

StudCourse(<u>StudNo,Major,CourseNo</u>,

Ctitle,InstrucName,InstructLocn,Grade)

- 2NF Remove partial key dependencies

Student(<u>StudNo</u>,StudName)

StudMajor(<u>StudNo,Major</u>,Advisor)

StudCourse(<u>StudNo,Major,CourseNo</u>,Grade)

Course(<u>CourseNo</u>,Ctitle,InstrucName,InstructLocn)

- 3NF Remove transitive dependencies

Student(<u>StudNo</u>,StudName)

StudMajor (StudNo, Major, Advisor)

StudCourse (StudNo, Major, CourseNo, Grade)

Course(CourseNo,Ctitle,InstrucName)

Instructor(InstructName,InstructLocn)

- BCNF Every determinant is a candidate key
- Student : only determinant is StudNo
- StudCourse: only determinant is StudNo,Major
- Course: only determinant is CourseNo
- Instructor: only determinant is InstrucName
- StudMajor: the determinants are
- StudNo,Major, or
- Adviser

Only StudNo,Major is a candidate key.

- BCNF

Student(StudNo,StudName)

StudCourse(StudNo,Major,CourseNo,Grade)

Course(CourseNo,Ctitle,InstrucName)

Instructor(InstructName,InstructLocn)

StudMajor(StudNo,Advisor)

Adviser(Adviser,Major)

{PRIVATE **}Fourth Normal Form (4NF) :** A relation that is in Boyce-Codd Normal Form and contains no nontrivial multi-valued dependencies.

Fourth normal form (4NF) is a stronger normal form than BCNF as it prevents relations from containing nontrivial MVDs, and hence data redundancy. The normalization of BCNF relations to 4NF involves the removal of the MVD from the relation by placing the attribute(s) in a new relation along with a copy of the determinant(s).

Self Assessment Exercise(s)

1. Why do databases have to be normalized?
2. What three data anomalies are likely to be the result of data redundancy?
3. How can such anomalies be eliminated?

Self Assessment Answer(s)

4.0 Conclusion

This unit discussed normalization process and the various normal form

5.0 Summary

You have learnt that

(i). A relation R is in first normal form (1NF) if and only if all underlying domains contain atomic values only

(ii). 2NF A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key

(iii). 3NF A relation R is in third normal form (3NF) if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key.

(iv). An attribute C is transitively dependent on attribute A if there exists an attribute B such that: AB and B C. Note that 3NF is concerned with transitive dependencies which do not involve candidate keys. A 3NF relation with more than one candidate key will clearly have transitive dependencies of the form: primary_key other_candidate_key any_non-key_column

(v). A relation R is in Boyce-Codd normal form (BCNF) if and only if every determinant is a candidate key

(vi). Insertion anomaly means that that some data cannot be inserted in the database.

(vii). Update anomaly means we have data redundancy in the database and to make any modification we have to change all copies of the redundant data or else the database will contain incorrect data..

(viii). Deletion anomaly means deleting some data cause other information to be lost

(ix). A functional dependency occurs when one attribute in a relation uniquely determines another attribute

6.0 Tutor Marked Assignment

1. Discuss the concept of Normalisation
2. Distinguish between functional, partial and transitive independency
3. How is the concept of functional dependency associated with the process of normalization?

Examine the table shown below.

branchNo	*branchAddress*	*telNos*
B001	3 Northwest avenue Bosso Minna	503-555-3618, 503-555-2727, 503-555-6534
B002	23 Ilupeju Estate Lagos	206-555-6756, 206-555-8836
B003	11 Bodija Steet Ibadan	212-371-3000

(a) Why is this table not in 1NF?

(b) Describe and illustrate the process of normalizing the data shown in this table to third normal form (3NF).

(c) Identify the primary, alternate and foreign keys in your 3NF relations.

7.0 References/Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://www.eng.mu.edu/corlissg/150.07f/ch05.html

http://www.visualbuilder.com/database/tutorial/concept-of-normalization/

http://databases.about.com/od/specificproducts/a/Database-Dependency.htm

http://web.eecs.umich.edu/~teorey/lec.notes.pdf

http://coronet.iicm.tugraz.at/Dbase1/scripts/rdbh04.htm

http://www.edrawsoft.com/chen-erd.php

http://ion.uwinnipeg.ca/~rmcfadye/2914/hypergraph/relationshipType.html

Module 3

Database Languages

Unit 1

Database Languages

Contents

1.0 Introduction

The main objective of a database management system is to allow its users to perform a number of operations on the database such as insert, delete, and retrieve data in abstract terms without knowing about the physical representations of data. To provide the various facilities to different types of users, a DBMS normally provides one or more specialized programming languages called **Database** (or **DBMS**) **Languages**.The DBMS mainly provides two database languages, namely, ***data definition language*** and ***data manipulation language*** to implement the databases. Data definition language (DDL) is used for defining the database schema. The DBMS comprises DDL compiler that identifies and stores the schema description in the DBMS catalog. Data manipulation language (DML) is used to express database queries and updates to manipulate the database.

2.0 Learning Outcomes

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

i. Explain Data Definition Language (DDL)
ii. Explain Data Manipulation Language (DML)
iii. Differentiate between DDL and DML
iv. Describe some commands used in DDL and DML

3.0. Learning Contents

3.1 Data Definition Language (DDL)

Database schema can be specified by a set of definitions expressed by a special language called a **data-definition language** (**DDL**).

In DBMSs where no strict separation between the levels of the database is maintained, the data definition language is used to define the conceptual and internal schemas for the database. On the other hand, in DBMSs, where a clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only. In such DBMSs, a separate language, namely, ***Storage Definition Language (SDL)*** is used to define the internal schema. Some of the DBMSs that are based on true three-schema architecture use a third language, namely, ***View Definition Language (VDL)*** to define the external schema.

The DDL statements are also used to specify the integrity rules (constraints) in order to maintain the integrity of the database. The various integrity constraints are domain constraints, referential integrity, assertions and authorization.. Like any other programming language, DDL also accepts input in the form of instructions (statements) and generates the description of schema as output. The output is placed in the Data Dictionary, which is a special type of table containing metadata that is data about data. The DBMS refers the data dictionary before reading or modifying the data. Note that the database users cannot update the data dictionary; instead it is only modified by database system itself.

For instance, the following statement in the SQL language defines the *account* table:

create table

account (*account-number* **char**(10),

balance **integer**)

Let's take a look at the structure and usage of four basic DDL commands:

CREATE

Installing a database management system (DBMS) on a computer allows you to create and manage many independent databases. For example, you may want to maintain a database of customer contacts for your sales department and a personnel database for your HR department. The CREATE command can be used to establish each of these databases on your platform. For example, the command:

CREATE DATABASE employees

creates an empty database named "employees" on your DBMS. After creating the database, your next step is to create tables that will contain data. Another variant of the CREATE command can be used for this purpose. The command:

CREATE TABLE personal_info (first_name char(20) not null, last_name char(20) not null, employee_id int not null) establishes a table titled "personal_info" in the current database. In our example, the table contains three attributes: first_name, last_name and employee_id. Don't worry about the other information included in the command -- we'll cover that in a future article.

USE

The USE command allows you to specify the database you wish to work with within your DBMS. For example, if we're currently working in the sales database and want to issue some commands that will affect the employees database, we would preface them with the following SQL command:

USE employees

It's important to always be conscious of the database you are working in before issuing SQL commands that manipulate data.

ALTER

Once you've created a table within a database, you may wish to modify the definition of it. The ALTER command allows you to make changes to the structure of a table without deleting and recreating it. Take a look at the following command:

```
ALTERTABLEpersonal_info
ADD salary money null
```

This example adds a new attribute to the personal_info table -- an employee's salary. The "money" argument specifies that an employee's salary will be stored using a dollars and cents format. Finally, the "null" keyword tells the database that it's OK for this field to contain no value for any given employee.

DROP

The final command of the Data Definition Language, DROP, allows us to remove entire database objects from our DBMS. For example, if we want to permanently remove the personal_info table that we created, we'd use the following command:

```
DROP TABLE personal_info
```

Similarly, the command below would be used to remove the entire employees database:

```
DROP DATABASE employees
```

Use this command with care! Remember that the DROP command removes entire data structures from your database. If you want to remove individual records, use the DELETE command of the Data Manipulation Language.

Self Assessment Exercise(s)

1. What are the basic DDL commands?

Self Assessment Answer(s)

3.2 Data Manipulation Language (DML)

Data Manipulation deals with the retrieval of information stored in the database, The insertion of new information into the database, The deletion of information from the database, The modification of information stored in the database

Once the database schemas are defined and the initial data is loaded into the database, several operations such as retrieval, insertion, deletion, and modification can be applied to the database. The DBMS provides data manipulation language (DML) that enables users to retrieve and manipulate the data. The statement which is used to retrieve the information is called a **query**. The part of the DML used to retrieve the information is called a **query language**. However, query language and DML are used synonymously though technically incorrect.

The DML are of two types, namely, *non-procedural DML* and *procedural DML*.

The **non-procedural** or **high-level** or **declarative DML** enables to specify the complex database operations concisely. It requires a user to specify *what* data is required without specifying *how* to retrieve the required data. For example, SQL (**Structured Query Language**) is a non-procedural query language as it enables user to easily define the structure or modify the data in the database without specifying the details of *how* to manipulate the database. The high-level DML statements can either be entered interactively or embedded in a general purpose programming language.

On the other hand, the **procedural** or **low-level DML** requires user to specify *what* data is required and *how* to access that data by providing step-by-step procedure. For example, relational algebra is procedural query language, which consists of set of operations such as select, project, union, etc., to manipulate the data in the database. Relational algebra and SQL are discussed in subsequent modules.

A brief look at the basic DML commands:

INSERT

The INSERT command in SQL is used to add records to an existing table. Returning to the personal_info example from the previous section, let's imagine that our HR department needs to add a new employee to their database. They could use a command similar to the one shown below:

```
INSERT INTO personal_info

values('bart','simpson',12345,$45000)
```

Note that there are four values specified for the record. These correspond to the table attributes in the order they were defined: first_name, last_name, employee_id, and salary.

SELECT

The SELECT command is the most commonly used command in SQL. It allows database users to retrieve the specific information they desire from an operational database. Let's take a look at a few examples, again using the personal_info table from

our employees database. The command shown below retrieves all of the information contained within the personal_info table. Note that the asterisk is used as a wildcard in SQL. This literally means "Select everything from the personal_info table."

```
SELECT *

FROM personal_info
```

Alternatively, users may want to limit the attributes that are retrieved from the database. For example, the Human Resources department may require a list of the last names of all employees in the company. The following SQL command would retrieve only that information:

```
SELECT last_name

FROM personal_info
```

Finally, the WHERE clause can be used to limit the records that are retrieved to those that meet specified criteria. The CEO might be interested in reviewing the personnel records of all highly paid employees. The following command retrieves all of the data contained within personal_info for records that have a salary value greater than N50,000:

```
SELECT *

FROM personal_info

WHERE salary >N50000
```

UPDATE

The UPDATE command can be used to modify information contained within a table, either in bulk or individually. Each year, our company gives all employees a 3% cost-of-living increase in their salary. The following SQL command could be used to quickly apply this to all of the employees stored in the database:

```
UPDATE personal_info

SET salary = salary * 1.03
```

On the other hand, our new employee Bart Simpson has demonstrated performance above and beyond the call of duty. Management wishes to recognize his stellar accomplishments with a $5,000 raise. The WHERE clause could be used to single out

Bart for this raise:

```
UPDATE personal_info

SET salary = salary + $5000

WHERE employee_id = 12345
```

DELETE

Finally, let's take a look at the DELETE command. You'll find that the syntax of this command is similar to that of the other DML commands. Unfortunately, our latest corporate earnings report didn't quite meet expectations and poor Bart has been laid off. The DELETE command with a WHERE clause can be used to remove his record

from the personal_info table:

```
DELETE FROM personal_info

WHERE employee_id = 12345
```

Self Assessment Exercise(s)

4. What are the basic DML command?

Self Assessment Answer(s)

4.0 Conclusion

This unit discusses the basic database language i.e. data manipulation language and data definition language

5.0 Summary

You have learnt:

1. DDL – the **data definition language**, used by the DBA and database designers to define the conceptual and internal schemas.
2. The DBMS has a DDL compiler to process DDL statements in order to identify the schema constructs, and to store the description in the catalogue.
3. In databases where there is a separation between the conceptual and internal schemas, DDL is used to specify the conceptual schema, and SDL, **storage definition language**, is used to specify the internal schema.
4. For a true three-schema architecture, VDL, **view definition language**, is used to specify the user views and their mappings to the conceptual schema. But in most DBMSs, the DDL is used to specify both the conceptual schema and the external schemas.
5. Once the schemas are compiled, and the database is populated with data, users need to manipulate the database. Manipulations include retrieval, insertion, deletion and modification.
6. The DBMS provides operations using the DML, **data manipulation language**.
7. In most DBMSs, the VDL, DML and the DML are not considered separate languages, but a comprehensive integrated language for conceptual schema definition, view definition and data manipulation. Storage definition is kept separate to fine-tune the performance, usually done by the DBA staff.
8. An example of a comprehensive language: SQL, which represents a VDL, DDL, DML as well as statements for constraint specification, etc.
9. DDL consists of really only 5 statements: CREATE, ALTER, DROP, GRANT, and REVOKE.
10. DDL statements are compiled, resulting in a set of tables stored in a special file called a **data dictionary** or **data directory**.
11. The data directory contains **metadata** (data about data).

6.0 Tutor Marked Assignment

Distinguish clearly between DDL and DML

7.0 References/Further Readings

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://www.cs.odu.edu/~ibl/450/pdf/view-on-line/dbarch/dbarch4.pdf

http://my.safaribooksonline.com/book

http://www.forkosh.com/sqltut.pdf

http://discovery.csc.ncsu.edu/Courses/csc742-S02/T12_TransacConcept_6.pdf

http://www.sis.pitt.edu/~valeriab/1022-spring08/Chapter15.pdf

http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter1/node16.html

http://www.risc.jku.at/education/courses/ws2003/is/ln.pdf

Unit 2

Relational Algebra

Contents

1.0 Introduction

The relational algebra is a *procedural* query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result. The fundamental operations in the relational algebra are *select, project*, *union*, *set difference*, *Cartesian product,* and *rename*. In addition to the fundamental operations, there are several other operations—namely, set intersection, natural join, division, and assignment. We will define these operations in terms of the fundamental operations.

2.0 Learning Outcomes

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

i. State the meaning of relational Algebra
ii. Describe the various operations on relational algebra
iii. Describe the various symbols used in relational algebra operations
iv. Write queries using relational symbols

3.0 Learning Contents

3.1 Basic Relational Algebra Operations

Basic set of relational model operations constitute the relational algebra. They are classified as follows.

Six fundamental operations: select (unary) , project (unary) , rename (unary) ,Cartesian product (binary) ,union (binary) ,set-difference (binary)

Several other operations, defined in terms of the fundamental operations: set-intersection, natural join, division, assignment, Operations produce a new relation as a result.

Fundamental Operations

1. **The Select Operation**

Select selects tuples that satisfy a given predicate. Select is denoted by a lowercase Greek sigma (σ), with the predicate appearing as a subscript. The argument relation is given in parentheses following the σ.

For example, to select tuples (rows) of the *borrow* relation where the branch is ``Minna'', we would write

$$\sigma_{bname=\text{"}SFU\text{"}}(borrow)$$

σ bname="MIN"(borrow)

Let Figure below be the *borrow* relation .
borrow relation:

bname	Lnum	Cname	amount
LAG	12	Johnson	1000
MIN	45	Seun	2000
ABJ	12	Smith	1500
SFU	123	Mohd	4000

branch relation

Name	Asset	bcity
SFU	1200000000	Suleja
MIN	1230000000	Abuja
LAG	5460000000	Ikeja

The new relation created as the result of this operation consists of one tuple:
(MIN,12,Seun,12300)

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

σ bname="Lagos" amount > 3400(borrow)

Suppose there is one more relation, *client*, with the table below scheme as

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

Cname	banker
Johnson	Johnson
Smith	Eunice

we might write

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

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

2 The Project Operation

Project copies its argument relation for the specified attributes only. Since a relation is a **set**, duplicate rows are eliminated.
Projection is denoted by the Greek capital letter pi (Π). The attributes to be copied appear as subscripts.

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

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

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

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

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

3 The Cartesian Product Operation

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

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

The result of $r_1 \times r_2$is a new relation with a tuple for each possible **pairing** of tuples from $r_1$and r_2.
In order to avoid ambiguity, the attribute names have attached to them the name of the relation from which they came. If no ambiguity will result, we drop the relation name.
The result $client \times customer$is a very large relation. If $r_1$has $n_1$tuples, and $r_2$has n_2 tuples, then $r = r_1 \times r_2$will have $n_1 n_2$tuples.
The resulting scheme is the concatenation of the schemes of $r_1$and r_2, with relation names added as mentioned.
To find the clients of banker John and the city in which they live, we need information in both *client* and *customer* relations. We can get this by writing

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

However, the *customer.cname* column contains customers of bankers other than Johnson. (Why?)
We want rows where *client.cname* = *customer.cname*. So we can write

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

to get just these tuples.

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

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

4 The Rename Operation

The **rename** operation solves the problems that occur with naming when performing the cartesian product of a relation with itself.
Suppose we want to find the names of all the customers who live on the same street and in the same city as Smith.
We can get the street and city of Smith by writing

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

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

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

where P is a selection predicate requiring *street* and *ccity* values to be equal.
Problem: how do we distinguish between the two street values appearing in the Cartesian product, as both come from a *customer* relation?
Solution: use the rename operator, denoted by the Greek letter rho (ρ).
We write $\rho_x(r)$
to get the relation r under the name of x.
If we use this to rename one of the two **customer** relations we are using, the ambiguities will disappear.

$$\Pi_{customer.cname}(\sigma_{cust2.street=customer.street \wedge cust2.ccity=customer.ccity}$$
$$(customer \times (\Pi_{street,ccity}(\sigma_{cname="Smith"}(\rho_{cust2}(customer))))))$$

5 The Union Operation

The **union** operation is denoted $\cup$ as in set theory. It returns the union (set union) of two compatible relations.
For a union operation $r \cup s$ to be legal, we require that
- r and s must have the same number of attributes.
- The domains of the corresponding attributes must be the same.

To find all customers of the SFU branch, we must find everyone who has a loan or an account or both at the branch.

We need both *borrow* and *deposit* relations for this:

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

As in all set operations, duplicates are eliminated, giving the relation of Figure

6. The Set Difference Operation

Set difference is denoted by the minus sign (_). It finds tuples that are in one relation, but not in another.
Thus $r - s$ results in a relation containing tuples that are in r but not in s.

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

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

We can do more with this operation. Suppose we want to find the largest account balance in the bank.
Strategy:

- Find a relation r containing the balances **not** the largest.
- Compute the set difference of r and the *deposit* relation.

To find r, we write

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

This resulting relation contains all balances except the largest one.

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

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

Additional Operations

Additional operations are defined in terms of the fundamental operations. They do not add power to the algebra, but are useful to simplify common queries.

1. **The Set Intersection Operation**

Set intersection is denoted by $\cap$, and returns a relation that contains tuples that are in **both** of its argument relations.
It does not add any power as

$$r \cap s = r - (r - s)$$

To find all customers having both a loan and an account at the SFU branch, we write

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

The Natural Join Operation

Often we want to simplify queries on a cartesian product. For example, to find all customers having a loan at the bank and the cities in which they live, we need *borrow* and *customer* relations:

$$\Pi_{borrow.cname,ccity}(\sigma_{borrow.cname=customer.cname}(borrow \times customer))$$

Our selection predicate obtains only those tuples pertaining to only one *cname*.
This type of operation is very common, so we have the **natural join**, denoted by a $\bowtie$ sign. Natural join combines a cartesian product and a selection into one operation. It

performs a selection forcing equality on those attributes that appear in both relation schemes. Duplicates are removed as in all relation operations.

To illustrate, we can rewrite the previous query as

$$\Pi_{cname,ccity}(borrow \bowtie customer)$$

We can now make a more formal definition of natural join.

- Consider $R$and $S$to be **sets** of attributes.
- We denote attributes appearing in **both** relations by $R \cap S$.
- We denote attributes in **either or both** relations by $R \cup S$.
- Consider two relations $r(R)$and $s(S)$.
- The natural join of r and s, denoted by $r \bowtie s$is a relation on scheme $R \cup S$.
- It is a projection onto $R \cup S$of a selection on $r \times s$where the predicate requires $r.A = s.A$for each attribute Ain $R \cap S$.

Formally,

$$r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1=s.A_1 \wedge r.A_2=s.A_2 \wedge \ldots \wedge r.A_n=s.A_n}(r \times s))$$

where $R \cap S = \{A_1, A_2, \ldots, A_n\}$.

To find the assets and names of all branches which have depositors living in Stamford, we need *customer*, *deposit* and *branch* relations:

$$\Pi_{bname,assets}\ (\sigma_{ccity=\text{"Stamford"}}\ (customer \bowtie deposit \bowtie branch))$$

Note that $\bowtie$is associative.

To find all customers who have both an account and a loan at the SFU branch:

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

This is equivalent to the set intersection version we wrote earlier. We see now that there can be several ways to write a query in the relational algebra.
If two relations $r(R)$and $s(S)$have no attributes in common, then $R \cap S = \emptyset$, and $r \bowtie s = r \times s$.

3 The Division Operation

Division, denoted$\div$, is suited to queries that include the phrase ``for all''.
Suppose we want to find all the customers who have an account at **all** branches located in Lagos.
Strategy: think of it as three steps.
We can obtain the names of all branches located in Brooklyn by

$r_1 = \pi$ bname(σ bcity="Lagos"(branch))

We can also find all *cname, bname* pairs for which the customer has an account by

$$r_2 = \Pi_{cname,bname}(deposit)$$

Now we need to find all customers who appear in r_2 with **every** branch name in r_1.

The divide operation provides exactly those customers:

π cname, bname(deposit)$\div$ π bname(σ city="Lagos"(branch))
which is simply $r_2 \div r_1$.

Formally,
a. Let $r(R)$ and $s(S)$ be relations.
b. Let $S \subseteq R$.
c. The relation $r \div s$ is a relation on scheme $R - S$.
d. A tuple t is in $r \div s$ if for every tuple t_s in s there is a tuple t_r in r satisfying both of the following:

$$t_r[S] = t_s[S] \qquad (3.2.1)$$

i. $$t_r[R-S] = t[R-S] \qquad (3.2.2)$$

e. These conditions say that the $R - S$ portion of a tuple t is in $r \div s$ if and only if there are tuples with the $r - s$ portion **and** the S portion in r for **every** value of the S portion in relation S.

f. We will look at this explanation in class more closely.
g. The division operation can be defined in terms of the fundamental operations.

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - r)$$

Read the text for a more detailed explanation.

The Assignment Operation

Sometimes it is useful to be able to write a relational algebra expression in parts using a temporary relation variable (as we did with r_1 and r_2 in the division example).
The assignment operation, denoted $\leftarrow$, works like assignment in a programming language.
We could rewrite our division definition as

$$temp \leftarrow \Pi_{R-S}(r)$$
$$temp - \Pi_{R-S}((temp \times s) - r)$$

No extra relation is added to the database, but the relation variable created can be used in subsequent expressions. Assignment to a permanent relation would constitute a modification to the database.

Self Assessment Exercise(s)

Given the database schema below.

employee (ename, street, city)

works (ename, cname, salary, jdate)

company (cname, city)

manages (ename, mname)

Write relational algebra expression

1. Find the names of all employees who work for First Bank Corporation
2. Find the names and cities of residence of all employees who work for First Bank Corporation
3. Find the names of all employees in this database who live in the same city as the company for which they work.

Self Assessment Answer(s)

4.0 Conclusion

The unit discusses relational algebra which uses a set of algebraic to operate on table so as to obtain results.

5.0 Summary

1. The **relational algebra** defines a set of algebraic operations that operate on tables, and output tables as their results. These operations can be combined to get expressions that express desired queries.
2. **Projection (p) -** The projection of a relation is defined as a projection of all its tuples over some set of attributes, i.e., it yields a *vertical subset* of the relation. It is used to either *reduce* the number of attributes (degree) in the resultant relation or to *reorder* attributes. The projection of a relation T on the attribute A is denoted by pA(T).
3. **Selection (s) -** Selects only some of the tuples, those satisfy given criteria, from the relation. It yields a *horizontal subset* of a given relation, i.e., the action is defined over a complete set of attribute names but only a subset of the tuples are included in the result.
4. **Union (È) -** Selects tuples that are in either P or Q or in both of them. *The*
   1. *duplicate tuples are eliminated.R = P È Q*
5. **Minus (–) -** Removes common tuples from the first relation.*R = P – Q*
6. **Cartesian Product or Cross Product (×) -** The cartesian product of two
7. relations is the concatenation of tuples belonging to the two relations and consisting of all possible combination of the tuples**.**

6.0 Tutor Marked Assignment

1. What are the fundamental operations used in Relational Algebra? What are the conditions for set (union, set-intersect and set-difference) operations in RA

2. Write the following queries using the relational algebra and show the result of each query as it applies to the tables below. You are not allowed to use any aggregates or the division operator.

S(SNUM, SNAME, STATUS, CITY) //Supplier relation

P(PNUM, PNAME, COLOR, WEIGHT, CITY) //Part relation

J(JNUM, JNAME, CITY) //Job relation

SPJ(SNUM, PNUM, JNUM, QTY) //Suppliers of Parts for Jobs relation

a. Find all suppliers with status ”50” that supply part ”P2”. For each supplier, output the name and the quantity of part ”P2” they supply.

b. Find and output the supplier numbers for all suppliers who either supply a ”Green” part or supply to a job in ”Abuja.”

3. Consider the following relational schema: **(7)**

PERSON (SSNUM, NAME, ADDRESS)

CAR (REGISTRATION_NUMBER, YEAR, MODEL)

ACCIDENT (DATE, DRIVER, CAR_REG_NO)

OWNS (SSNUM, LICENSE)

Construct the following relational algebra queries:

(i) Find the names of persons who are involved in an accident.

(ii) Find the registration number of cars which were not involved in any repair.

7.0 References/Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter3/node7.html

http://pheatt.emporia.edu/courses/2012/cs444f12/hand08c/hand08c.pdf http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter3/node7.html

www.databasteknik.se/webbkursen/relalg-lecture/index.html

www.cise.ufl.edu/class/cis4301sp07/notes/notes_13.ppt

Unit 3

Relational Calculus

Contents

1.0 Introduction

Relational calculus is an alternative to relational algebra. In contrast to the algebra, which is procedural, the calculus is nonprocedural, or *declarative*, in that it allows us to describe the set of answers without being explicit about how they should be computed. Relational calculus has had a big influence on the design of commercial query languages such as SQL and, especially, Query-by-Example (QBE).

The variant of the calculus that we present in detail is called the tuple relational calculus (TRC). Variables in TRC take on tuples as values. In another variant, called the domain relational calculus (DRC), the variables range over field values. TRC has had more of an influence on SQL, while DRC has strongly influenced QBE.

2.0 Learning Outcomes

At the end of the study you should be able to:

i. State the meaning of relational calculus
ii. Explain the differences between relational algebra and relational calculus
iii. Write simple queries using relational calculus

3.0 Learning Contents

3.1 The Relational Calculus

The relational calculus is an alternative to the relational algebra as a basis for query languages. It is derived from predicate calculus.

1. A predicate is an assertion that we require to be true. When we formulate a query in the relational calculus, we specify a predicate that the object(s) we are looking for must satisfy.
2. Unlike relational algebra - which is procedural - relational calculus is non-procedural - i.e. we specify what requirements the result must satisfy, not how to compute it. Just as the relational algebra serves as the mathematical foundation for the commercial query language SQL, the relational calculus serves as the mathematical foundation for various commercial visual query languages. There are two variants of the relational calculus: the tuple relational calculus and the domain relational calculus.

In the tuple relational calculus, variables represent tuples, and predicates are formulated in terms of attributes of a tuple variable.

Ex: Find book tuples for which author = dog:

{ t | t ε book ^ t[author] = dog }

2. In the domain relational calculus, variables represent individual attributes, and complete tuples are represented as lists of attributes.

Ex: The above query:

{ <call_number, title, author> |

<call_number, title, author> ε book ^ author = dog }

In either case, we represent a query by a predicate which we want the result to satisfy.

For example let us look at examples of relational calculus equivalents to each basic relational algebra operation.

1. Relational algebra SELECTION:

σ book author = dog

See the two examples above

2. Relational algebra PROJECTION:

π borrower

last_name

first_name

a) tuple r.c.:

{ t | ∃ s (s ε borrower ^ t[last_name]=s[last_name] ^t[first_name]=s[first_name]) }

where t is on the new scheme (last_name, first_name)

b) domain r.c.:{ <last_name, first_name> | ∃ borrower_id(< borrower_id, last_name, first_name> ε borrower) }

3. Relational algebra NATURAL JOIN: checked_out |X| borrower

a) i. tuple r.c.:

{ t | ∃ u, v (u ε checked_out ^ v ε borrower ^u[borrower_id] = v[borrower_id] ^

t[borrower_id] = u[borrower_id] ^t[call_number] = u[call_number] ^

t[date_due] = u[date_due] ^t[last_name] = v[last_name] ^t[first_name] = v[first_name]) }

where t is on the new scheme

(borrower_id, call_number, date_due, last_name, first_name)

b) domain r.c.:

{ <borrower_id, call_number, date_due, last_name, first_name> |

<borrower_id, call_number, date_due> ε checked_out ^<borrower_id, last_name, first_name> ε borrower }

4. Relational algebra UNION: Suppose we have two tables that have the same scheme - say student_borrower and fac_staff_borrower – and want a table including all persons in either group:

a) tuple r.c.:

{ t | (t ε student_borrower) v (t ε fac_staff_borrower) }

b) domain r.c.:

{ <borrower_id, last_name, first_name> |(<borrower_id, last_name, first_name> ε student_borrower) v(<borrower_id, last_name, first_name> ε fac_staff_borrower)}

5. Relational algebra DIFFERENCE: Suppose, instead, we have a general borrower table - which contains information on all borrowers - plus astudent_borrower table, which contains only student borrowers, and we want to list borrowers who are not students. In relational algebra, this would be borrower - student_borrower

a) tuple r.c.:

{ t | (t ε borrower) ^ ¬ (t ε student_borrower) }

b) domain r.c.:

{ <borrower_id, last_name, first_name> |(<borrower_id, last_name, first_name> ε borrower) ^¬ (<borrower_id, last_name, first_name> ε student_borrower)}

6. A more complete example:

Find the last name of all borrowers who have an overdue bookπ σ checked_out |X| borrowerlast_name date_due <-- whatever today is --

Ask class to do in each relational calculus:

tuple:

{ t | ∃ u,v(u ε checked_out ^ v ε borrower ^t[last_name] = v[last_name] ^

u[borrower_id] = v[borrower_id] ^u[date_due] < "September 18, 2002") }

domain:

{ <last_name> | ∃ borrower_id, call_number, date_due, first_name

(<borrower_id, call_number, date_due> ε checked_out ^<borrower_id, last_name, first_name> ε borrower ^date_due < "September 18, 2002") }

F. One important property of relational calculus formulas is SAFETY. A relational calculus formula is said to be safe iff it does not require us to inspect infinitely many objects.

1. Example: the query { t | ¬ (t ε R) } is not safe. For any relation R, there are infinitely many tuples that are not members of that relation!

2. Unsafe formulas are most often the result of improper use of not. In general, the set of all values NOT satisfying some predicate is infinite; so when not is used it must be coupled with an additional condition that narrows the scope of consideration - e.g. if P(x) and Q(x) are safeformulas then

¬ Q(x)is unsafe, but

P(x) ^ ¬ Q(x)is safe.

3. The most common way to ensure that a formula is safe is to include a formula that restricts consideration to tuples from a particular relation -

i.e. a formula of the form t ε R or <a,b,c> ε R.

4. Notice that the issue of safety is unique to the relational calculus. One cannot formulate an unsafe query in the relational algebra - hence only safe relational calculus formulas have relational algebra equivalents.

1. Example : RDC query to retrieve names of all professors who have taught MGT123:

$$\begin{aligned}\{N \mid\ & \exists I \in Professor.Id\ \exists D \in Professor.DeptId \\ & (Professor(I,N,D)\ AND\ \exists S \in Teaching.Semester \\ & (Teaching(I, MGT123, S)))\}\end{aligned}$$

This can be abbreviated:

$$\{N \mid\ Professor(I,N,D)\ AND\ (Teaching(I, MGT123, S))\}$$

2 Retrieve the birthdate and SIN of the employee(s) whose last name is ”Smith”

and whose age is less than 30

{t.BDATE, t.SIN **|** EMPLOYEE(t) and t.LNAME = "Smith" and t.AGE < 30}

Self Assessment Exercise(s)

1 Consider the following relation schemes:

Project (Project#, Project_name, chief_architect)

Employee (Emp#, Empname)

Assigned_To (Project#, Emp#)

Give expression in Tuple calculus and Domain calculus for each of the queries below:

(i) Get the employee numbers of employees who work on all projects.

(ii) Get the employee numbers of employees who do not work on the COMP123 project.

2. You are given the following relational schema for students.

Student(SID, Sname, Curriculum)

Takes (CourseID, SID, Semester, Grade) //Semester is of the form ”Fall 98”, ”Spring 99”, etc.

Write the following queries in tuple relational calculus:

a. Find the id of all students who took the same course in two different semesters.

b. Find the name of all students who never got an F.

Self Assessment Answer(s)

4.0 Conclusion

This unit discusses relational calculus which is also used in querying database. The two variants are tuple relational calculus and domain relational calculus.

5.0 Summary

You have learnt that:

1. Relational calculus is an alternative to relational algebra.
2. In contrast to the algebra, which is procedural, the calculus is nonprocedural, or *declarative*, in that it allows us to describe the set of answers without being explicit about how they should be computed.
3. Relational calculus has had a big influence on the design of commercial query languages such as SQL and, especially, Query-by-Example (QBE).

6.0 Tutor-Marked Assignment

1 Consider the following relational schema:

Doctor(DName,Reg_no)

Patient(Pname, Disease)

Assigned_To (Pname,Dname)

Give expression in both Tuple calculus and Domain calculus for each of the queries:

(i) Get the names of patients who are assigned to more than one doctor.

(ii) Get the names of doctors who are treating patients with ‘Polio’.

2 Given the following relational schema

EMPLOYEE(ssn, firstname, lastname, title, salary, dept id, mgr id)

DEPARTMENT(dept id, departmentname, totalbudget)

Write the following queries in tuple relational calculus:

1. Find the first and last name of managers such that none of his/her employees work in the same department as their boss (i.e. the department of all employees is different than the department of their boss).

2. Find the name of all departments that has no employee working in them.

7.0 References/ Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://sauron.wlu.ca/physcomp/ikotsireas/TupleCalculus.pdf

http://www.cse.ohio-state.edu/~gurari/course/cse670/html/cse670Ch15.html

http://www.math-cs.gordon.edu/courses/cps352/lectures-2008/RelationalCalculus.pdf

http://coronet.iicm.tugraz.at/Dbase1/scripts/rdbh07.htm

Unit 4

Introduction to Structural Query Language

Contents

1.0 Introduction

SQL stands for Structured Query Language, and it is a very powerful and diverse language used to create and query databases. It is pronounced as **sequel** and is not a conventional computer programming language in the normal sense of the phrase. Instead SQL is a language used exclusively to create, manipulate and interrogate databases. SQL is about data and results; each SQL statement returns a result, whether that result be a query, an update to a record or the creation of a database table. This unit gives introduction to the basic concept of SQL.

2.0 Learning Outcomes

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

i. State the meaning of SQL
ii. Discuss the history of SQL
iii. Mention the parts of SQL
iv. Mention the various types of SQL
v. State the advantages and disadvantages of SQL

3.0 Learning Contents

3.1 Structural Query Language

SQL Introduction

SQL stands for "Structured Query Language" and can be pronounced as "SQL" or "sequel – (Structured English Query Language)". It is a query language used for accessing and modifying information in the database. IBM first developed SQL in 1970s. Also it is an ANSI/ISO standard. It has become a Standard Universal Language used by most of the relational database management systems (RDBMS). Some of the RDBMS systems are: Oracle, Microsoft SQL server, Sybase etc. Most of these have provided their own implementation thus enhancing it's feature and making it a powerful tool. Few of the sql commands used in sql programming are SELECT Statement, UPDATE Statement, INSERT INTO Statement, DELETE Statement, WHERE Clause, ORDER BY Clause, GROUP BY Clause, ORDER Clause, Joins, Views, GROUP Functions, Indexes etc.
In a simple manner, SQL is a non-procedural, English-like language that processes data in groups of records rather than one record at a time.

Few functions of SQL are:

i. store data
ii. modify data
iii. retrieve data
iv. modify data
v. delete data
vi. create tables and other database objects

vii. delete data

Self Assessment Exercise(s)

1. Mention four(4) SQL Commands?

Self Assessment Answer(s)

3.2 Parts of SQL

The SQL language has several parts:

i. **Data-definition language** (DDL). The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.
ii. **Interactive data-manipulation language** (DML). The SQL DML includes a query language based on both the relational algebra and the tuple relational calculus. It includes also commands to insert tuples into, delete tuples from, and modify tuples in the database.
iii. **View definition**. The SQL DDL includes commands for defining views.
iv. **Transaction control**. SQL includes commands for specifying the beginning and ending of transactions.
v. **Embedded SQL** and **dynamic SQL**. Embedded and dynamic SQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, Java, PL/I, Cobol, Pascal, and Fortran.
vi. **Integrity**. The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are disallowed.
vii. **Authorization**. The SQL DDL includes commands for specifying access rights to relations and views.
viii. **Security:** SQL provides mechanisms to control users' access to data objects such as tables and views.
ix. **Transaction management:** Various commands allow a user to explicitly control aspects of how a transaction is to be executed.
x. **Client-server execution and remote database access:** These commands control how a *client* application program can connect to an SQL database *server*, or access data from a database over a network.

Self Assessment Exercise(s)

1. The part of SQL that deals with provison of mechanism to control users' access to data object is?
2. The part that incorporates the use of query language is?

Self Assessment Answer(s)

3.3 Description of Popular RDBMS

There are many popular RDBMS available to work with. This unit gives a brief overview of few most popular RDBMS. This would help you to compare their basic features:

MySQL

MySQL is open source SQL database, which is developed by Swedish company MySQL AB. MySQL is pronounced "my ess-que-ell," in contrast with SQL, pronounced "sequel."MySQL is supporting many different platforms including Microsoft Windows, the major Linux distributions, UNIX, and Mac OS X.
MySQL has free and paid versions, depending on its usage (non-commercial/commercial) and features. MySQL comes with a very fast, multi-threaded, multi-user, and robust SQL database server.

History:

- Development of MySQL by Michael Widenius & David Axmark beginning in 1994.
- First internal release on 23 May 1995.
- Windows version was released on 8 January 1998 for Windows 95 and NT.
- Version 3.23: beta from June 2000, production release January 2001.
- Version 4.0: beta from August 2002, production release March 2003 (unions).
- Version 4.01: beta from August 2003, Jyoti adopts MySQL for database tracking.
- Version 4.1: beta from June 2004, production release October 2004.
- Version 5.0: beta from March 2005, production release October 2005.
- Sun Microsystems acquired MySQL AB on 26 February 2008.
- Version 5.1: production release 27 November 2008.

Features:

The features are High Performance, high Availability, Scalability and Flexibility Run anything, Robust Transactional Support, Web and Data Warehouse Strengths, Strong Data Protection, Comprehensive Application Development, Management Ease, Open Source Freedom and 24 x 7 Support, Lowest Total Cost of Ownership.

MS SQL Server

MS SQL Server is a Relational Database Management System developed by Microsoft Inc. Its primary query languages are:T-SQL and ANSI SQL.

History:

- 1987 - Sybase releases SQL Server for UNIX.
- 1988 - Microsoft, Sybase, and Aston-Tate port SQL Server to OS/2.
- 1989 - Microsoft, Sybase, and Aston-Tate release SQL Server 1.0 for OS/2.

- 1990 - SQL Server 1.1 is released with support for Windows 3.0 clients.
- Aston-Tate drops out of SQL Server development.
- 2000 - Microsoft releases SQL Server 2000.
- 2001 - Microsoft releases XML for SQL Server Web Release 1 (download).
- 2002 - Microsoft releases SQLXML 2.0 (renamed from XML for SQL Server).
- 2002 - Microsoft releases SQLXML 3.0.
- 2005 - Microsoft releases SQL Server 2005 on November 7th, 2005.
- MySQL Server 5.5 is currently generally available (as of December 2010[update]
- MySQL Server 6.0.11-alpha was announced[77] on May 22, 2009 as the last release of the 6.0 line. Future MySQL Server development uses a New Release Model. Features developed for 6.0 are being incorporated into future releases.
- MySQL 5.6, a development milestone release, was announced at the MySQL users conference 2011.

Features:

The features are High Performance, High Availability, Database mirroring, Database snapshots, CLR integration, Service Broker, DDL triggers, Ranking functions,Row version-based isolation levels, XML integration,TRY...CATCH, Database Mail. Semi synchronous replication, SIGNAL and RESIGNAL statement in compliance with the SQL standard, Support for supplementary Unicode character sets utf16, utf32, and utf8mb4, New options for user-defined partitioning.

Oracle

It is very large and multi-user database management system. Oracle is a relational database management system developed by 'Oracle Corporation'.Oracle works to efficiently manage its resource, a database of information, among the multiple clients requesting and sending data in the network.It is an excellent database server choice for client/server computing. Oracle supports all major operating systems for both clients and servers, including MSDOS, NetWare, UnixWare, OS/2 and most UNIX flavors.

History:

Oracle began in 1977 and celebrating its 32 wonderful years in the industry (from 1977 to 2009).

- 1977 - Larry Ellison, Bob Miner and Ed Oates founded Software Development Laboratories to undertake development work.
- 1979 - Version 2.0 of Oracle was released and it became first commercial relational database and first SQL database. The company changed its name to Relational Software Inc. (RSI).
- 1981 - RSI started developing tools for Oracle.
- 1982 - RSI was renamed to Oracle Corporation.
- 1983 - Oracle released version 3.0, rewritten in C language and ran on multiple platforms.

- 1984 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency etc.
- 1985 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency etc.
- 2007 - Oracle has released Oracle11g. The new version focused on better partitioning, easy migration etc.

Features includes Concurrency, Read Consistency, Locking Mechanisms, Quiesce Database, Portability, Self managing database, SQL*Plus, Scheduler, Resource Manager, Data Warehousing, Materialized views. Bitmap indexes, Table compression, Parallel Execution, Analytic SQL, Data mining, Partitioning.

Ms- Access

This is one of the most popular Microsoft products. Microsoft Access is entry-level database management software. MS Access database is not only an inexpensive but also powerful database for small-scale projects.MS Access uses the Jet database engine which utilizes a specific SQL language dialect (sometimes referred to as Jet SQL).MS Access comes with the professional edition of MS Office package. MS Access has easy to use intuitive graphical interface.

History:

- 1992 - Access version 1.0 was released.
- 1993 - Access 1.1 release to improve compatibility with include the Access Basic programming language.
- The most significant transition was from the Access 97 to the Access 2000
- 2007 - Access 2007, a new database format was introduced ACCDB which supports complex data types such as multi valued and attachment fields.
- 2010- With Access 2010, a new version of the ACCDB format supports hosting on a SharePoint 2010 server for exposure to the web

Features includes

i. Users can create tables, queries, forms and reports, and connect them together with macros.
ii. The import and export of data to many formats including Excel, Outlook, ASCII, dBase, Paradox, FoxPro, SQL Server, Oracle, ODBC, etc.
iii. There is also the Jet Database format (MDB or ACCDB in Access 2007) which can contain the application and data in one file. This makes it very convenient to distribute the entire application to another user, who can run it in disconnected environments.
iv. Microsoft Access offers parameterized queries. These queries and Access tables can be referenced from other programs like VB6 and .NET through DAO or ADO.
v. The desktop editions of Microsoft SQL Server can be used with Access as an alternative to the Jet Database Engine.

vi. Microsoft Access is a file server-based database. Unlike client-server relational database management systems (RDBMS), Microsoft Access does not implement database triggers, stored procedures, or transaction logging.

Self Assessment Exercise(s)

1. Mention four (4) popular RDMS?

Self Assessment Answer(s)

Advantages of SQL

i. High Speed: SQL Queries can be used to retrieve large amounts of records from a database quickly and efficiently.

ii. Well Defined Standards Exist: SQL databases use long-established standard which is being adopted by ANSI & ISO. Non-SQLdatabases do not adhere to any clear standard.

iii. No Coding Required: Using standard SQL it is easier to manage database systems without having to write substantial amount of code.

iv. Emergence of ORDBMS: Previously SQL databases were synonymous with relational database. With the emergence of Object-oriented DBMS, object storage capabilities are extended to relational databases.

Disadvantages of SQL

i. Difficulty in Interfacing: Interfacing an SQL database is more complex than adding a few lines of code.

ii. More Features Implemented in Proprietary way: Although SQL databases conform to ANSI & ISO standards, some databases go for proprietary extensions to standard SQL to ensure vendor lock-in

4.0 Conclusion

This unit gives a brief introduction to SQL and compares different RDBMS

5.0 Summary

You have learnt:

i. SQL stands for Structured Query Language,

ii. It is a very powerful and diverse language used to create and query databases

iii. SQL is a non-procedural, English-like language that processes data in groups of records rather than one record at a time.

iv. Few functions of SQL are to store, modify, retrieve, modify, and delete data as well as to create tables and other database objects

6.0 Tutor-Marked Assignment

1 Explain the various parts of SQL

2 what are the advantages and disadvantages of SQL

3 Compare and contrast MySQLOracle, Microsoft Server SQL and MS access

7.0 References/Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://www.tutorialspoint.com/sql/sql-syntax.htm

http://en.wikipedia.org/wiki/Oracle_Database

http://en.wikipedia.org/wiki/Microsoft_Access

http://alakroy.ueuo.com/nita10/SQL_Tutorial_DBMS_Lab.pdf

http://www.cs.iit.edu/~cs561/cs425/VenkatashSQLIntro/Advantages%20&%20Disadvantages.html

Unit 5

Structural Query Language (SQL) Commands

Contents

1.0 Introduction

SQL is a query language that allows access to data residing in relational database management systems (RDBMS), such as Sybase, Oracle, Informix, DB2, Microsoft SQL Server, Access and many others. To retrieve information users execute *'queries'* to pull the requested information from the database using criteria that is defined by the user. A query, in its simplest form is constructed using the following basic query statements SELECT, FROM, WHERE and ORDER BY. This unit discusses some common SQL commands that can work across multiple database applications

2.0 Learning Outcomes

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

i. Explain the basic SQL commands
ii. Describe how to create tables
iii. Formulate queries using SQL commands
iv. Use aggregate functions

3.0 Learning Contents

3.1 SQL Commands

SQL commands are lines of SQL code that ask the SQL application to perform simple tasks against with data inside of a database. Often, we refer to commands as query statements or scripts; all of these terms are synonymous. SQL commands are declarative sentences or 'orders' executed against a SQL database. The typical command is comprised of several different components including *clauses, functions, expressions, or objects* but the only required components are a SQL Clause and the data object (a database or a database table).

These are the 16 SQL commands, separated into commonly used groups:

The Data Manipulation Language (DML) commands:

1. SELECT
2. INSERT
3. UPDATE
4. DELETE
5. MERGE

The Data Definition Language (DDL) commands:

1. CREATE
2. ALTER
3. DROP
4. RENAME
5. TRUNCATE
6. COMMENT

The Data Control Language (DCL) commands:

1. GRANT

2 REVOKE

The Transaction Control Language (TCL) commands:

1. COMMIT

2. ROLLBACK

3.SAVEPOINT

Creating Tables

You must create your tables before you can enter data into them. Use the **Create Table** command.

Syntax:

Create table *tablename* using *filename*

(*fieldnamefieldtype*(*length*),
fieldnamefieldtype(*length*),
fieldnamefieldtype(*length*));

i. Table names cannot exceed 20 characters.
ii. Table names must be unique within a database.
iii. Field names must be unique within a table.
iv. You may specify the data file to use. If you do not specify a data file, Scalable SQL will create one, using a .dat extension.
v. The list of fields must be enclosed in parentheses.
vi. You must specify the field type.

Examples:
Char -- a character string

Float -- a number

Date -- a date field

Logical -- a logical field

i. You must specify the field length.
ii. The field length must be enclosed in parentheses.
iii. You must separate field definitions with commas.
iv. You must end each SQL statement with a semicolon.

Example: To create a table called family with the following column FAM_ID, NAME_LAST, STR_ADD, CITY, STATE, MAR_STA

We will use the command:

```
Create table FAMILY
(FAM_ID char(10) primary key,
LAST_NAME char(30),
STR_ADD char(30),
CITY  char(15),
STATEchar(20),

AGE int,
MAR_STA char(10));
```

SQL INSERT statement

You can import data into SQL from another data source by using the **Insert** command. To import data into the table we created above we use the following command. The INSERT INTO statement is used to insert new row into the table

Syntax:
Insert into *tablename*
(*fieldname, fieldname, fieldname*)
Values
(*@fieldname, @fieldname, @fieldname*);
For example to add record to our FAMILY table we issue the command

```
INSERT INTO FAMILY(FAM_ID, LAST_NAME, STR_ADD,CITY,STATE,AGE)
VALUES('F100','Adepoju','Tunga','Niger',20)
```

SQL DELETE statement
It is used delete rows in a table

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

```
E,g DELETE FROM FAMILY
        WHERE LAST_NAME='John'
```

Delete All Rows

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

```
DELETE FROM table_name
Or
DELETE * FROM table_name
```

The UPDATE Statement

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

SQL UPDATE Syntax

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

SQL SELECT Clause

Remember that the minimum requirements for writing a SQL query is the SELECT & FROM clauses. The SELECT clause is where you request the pieces of information, *the columns* (separated by commas), that you want to see, and the FROM clause is where you define the table or tables from which the columns reside in.

The basic form of an SQL query is as follows:

SELECT [DISTINCT]**select-list**

FROM **from-list**

WHERE **qualification**

For example:
SELECT *column_name, column_name, column_name ...*
FROM *table_name*

Example: if we wanted to retrieve a list of all families and their addresses from family table created under create command, the query would look like this;

```
SELECT NAME_LAST, STR_ADD, CITY, STATE
FROM FAMILY;
```

The results of the query would be record showing LAST_NAME, STR_ADD, CITY and STATE

Note that we did not request to see the FAM_ID and MAR_STA columns specifically we asked to see the *columns* called NAME_LAST, STR_ADD, CITY and STATE

Each column and table name must be specified exactly as it is defined and will not contain any spaces.

If you wanted to see every column without having to type each column name you could use an asterisk (*);

SELECT *
FROM *table_name*
Or in our example:

```
SELECT *
FROM FAMILY;
```
this will list all record in the family table

SQL** WHERE **Clause

The *WHERE* clause is used to conditionally retrieve only the information that you want to display. This will limit the number of rows that answer the query and are fetched. In many cases, this is where most of the "action" of a query takes place. Here is a list of the operators that can be used. It is important to note that when querying character or date/date time type of data that the parameters used in the WHERE clause be surrounded with an apostrophe *(')*, when querying against numeric type data it is not required.

Table 1 operators

Operators	Description
=	Equal
<> or! =	Not equal
<	Less than
>	Greater than
<=	Less or equal to
>=	Greater than or Equal To
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

To see how the WHERE clause works lets take a look at some examples.

If we wanted to see a list of individuals that were married you could run the following query;

```
SELECT LAST_NAME, STATE
FROM FAMILY
WHERE MARITAL_STATUS = 'Married';
```

This will display only the last name and state of married person from the family table

Order By

The *ORDER BY* clause is used to sort the result set in the desired sequence. The syntax for order by is;

```
ORDER BY COLUMNNAME [ASC/DESC]
```

The following example sorts the result in descending first name within ascending last name order. The default for the ORDER BY clause is ascending order so if you want your results to be in ascending order it is not necessary to specify the ASC parameter.

```
SELECT LAST_NAME, STATE
FROM FAMILY
WHERE MARITAL_STATUS = 'Married';
ORDER BY LAST_NAME ASC
```

Compound Conditions

The ***AND*** operator joins two or more conditions and displays a row only if **ALL** of the conditions are met.

For example, building upon our query above, we may want to further restrict the query results using the date of birth column. If we only want to retrieve Individuals that are married and have age less than 34.

```
SELECT LAST_NAME,STATE
FROM FAMILY
WHERE MAR_STAT = 'Married' AND AGE <= '34';
```

Using LIKE

The LIKE operator can be used to search a column at a somewhat higher level using special wildcard characters. For instance, if you want to return records where a column value begins or ends with a certain character or set of characters.

If you wanted to find all of the individuals whose last name began with an 'S' you could write a query like this;

```
 SELECT LAST_NAME, CITY
FROM INDIVIDUAL
WHERE LAST_NAME LIKE 'S%'
```

The various wildcard that can be used are depicted in the table below:

Table 2 wildcard

Wildcard	**Description**
_ (underscore)	matches any single character
%	matches a string of one or more characters
[ ]	matches any single character within the specified range (e.g. [a-f]) or set (e.g. [abcdef]).

[^]	matches any single character not within the specified range (e.g. [^a-f]) or set (e.g. [^abcdef]).

A few examples should help clarify these rules.

- **WHERE** FirstName **LIKE** '_im' finds all three-letter first names that end with 'im' (e.g. Jim, Tim).
- **WHERE** LastName **LIKE** '%ade' finds all employees whose last name ends with 'ade'
- **WHERE** LastName **LIKE** '%ade%' finds all employees whose last name includes 'ade' anywhere in the name.
- **WHERE** FirstName **LIKE** '[JT]im' finds three-letter first names that end with 'im' and begin with either 'J' or 'T' (that is, *only* Jim and Tim)
- **WHERE** LastName **LIKE** 'm[^c]%' finds all last names beginning with 'm' where the following (second) letter is not 'c'.

Database View

A view in SQL terminology is a single table that is derived from other tables. These other tables could be base tables or previously defined views. A view does not necessarily exist in physical form; it is considered a virtual table, in contrast to base tables, whose tuples are actually stored in the database. This limits the possible update operations that can be applied to views, but it does not provide any limitations on querying a view. A view represents a different perspective of a base relation(s). The definition of a view in a create view statement is stored in the system catalog. Any attribute in the view can be updated as long as the attribute is simple and not derived from a computation involving two or more base relation attribute. View that involve a join may or may not be updatable. Such views are not updatable if they do not include the primary keys of the base relations.

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

SQL CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

Example 1: Consider the relations defined below:

PHYSICIAN (regno, name, telno, city)

PATIENT (pname, street, city)

VISIT (pname, regno, date_of_visit, fee)

Where the regno and pname identify the physician and the patient uniquely respectively.

Express queries (i) to (iii) in SQL.

i. Get the name and regno of physicians who are in Minna.
ii. Find the name and city of patient(s) who visited a physician on 31 August
iii. 2012.
iv. Get the name of the physician and the total number of patients who havevisited her.
v. What does the following SQL query answer

SELECT DISTINCT name

FROM PHYSICIAN P

WHERE NOT EXISTS

(SELECT *

FROM VISIT

WHERE regno = p.regno)

Answer

(i) Select name, regno from PHYSICIAN where city = 'Minna';

(ii) Select pname, city from PATIENT,VISIT where PATIENT.pname=VISIT.pname

and date_of_visit = '31-Aug-12';

(iii) select name, count(*) from PHYSICIAN, VISIT

where PHYSICIAN.regno = VISIT.regno group by

Physician. regno;

(iv) This will give the name of physicians who have not visited any patient.

Self Assessment Exercise(s)

Consider the following relational database:

STUDENT (name, student#, class, major)

COURSE (course name, course#, credit hours, department)

SECTION (section identifier, course#, semester, year, instructor)

GRADE_REPORT (student#, section identifier, grade)

PREREQUISITE (course#, presequisite#)

Specify the following queries in SQL on the above database schema.

(i) Retrieve the names of all students majoring in 'CS' (Computer Science).

(ii) Retrieve the names of all courses taught by Professor John in 2012

(iii) Delete the record for the student whose name is 'Smith' and whose student number is 17.

(iv) Insert a new course <'Knowledge Engineering', 'CS4390', 3, 'CS'>

Self Assessment Answer(s)

4.0 Conclusion

This unit discusses the basic SQL command like CREATE, SELECT, INSERT, DELETE, UPDATE and some relational operators

5.0 Summary

You have learnt:

1. SQL is a query language that allows access to data residing in relational database management systems (RDBMS), such as Sybase, Oracle, Informix, DB2, Microsoft SQL Server, Access and many others.
2. To retrieve information users execute *'queries'* to pull the requested information from the database using criteria that is defined by the user.
3. A query, in its simplest form is constructed using the following basic query statements SELECT, FROM, WHERE and ORDER BY
4. CREATE is used to create a new table
5. INSERT is to add records into the table
6. The SELECT clause is where you request the pieces of information
7. Operators are also used in SQL commands
8. A view in SQL terminology is a single table that is derived from other tables

6.0 Tutor Marked Assignment

1. Given the database schema below

employee (ename, street, city)

works (ename, cname, salary, jdate)

company (cname, city)

manages (ename, mname)

Give the corresponding SQL expression for the following queries

a. Find the names of all employees who work for First Bank Corporation.

b. Find the names and cities of residence of all employees who work for First

c. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than Tk. 30000.

d. Find names, street addresses and cities of residence of all employees who work under manager Sabbir and who joined before January 01, 2009.

e. Find the names of the employees living in the same city where Rahim is residing.

f. Find the names of all employees in this database who do not work for First Bank Corporation.

2 What is a view in SQL?

7.0 References/Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://www.baycongroup.com/tocsql.htm

http://www.stat.berkeley.edu/~spector/sql.pdf

http://www.thundersoftware.com/churchdb/sqltutorial/sql_tutorial.html

http://www.w3schools.com/sql/sql_where.asp

http://oreilly.com/catalog/sqlnut/chapter/ch04.html

http://infolab.stanford.edu/~ullman/fcdb/slides/slides6.pdf

Module 4

Other Topics

Unit 1

Transaction Processing

Contents

1.0 Introduction

Transaction processing is designed to maintain a database Integrity (typically a database or some modern file systems) in a known, consistent state, by ensuring that any operations carried out on the system that are interdependent are either all completed successfully or all canceled successfully.

Transaction processing allows multiple individual operations to be linked together automatically as a single, indivisible transaction. The transaction-processing system ensures that either all operations in a transaction are completed without error, or none of them are. If some of the operations are completed but errors occur when the others are attempted, the transaction-processing system "rolls back" *all* of the operations of the transaction (including the successful ones), thereby erasing all traces of the transaction and restoring the system to the consistent, known state that it was in before processing of the transaction began. If all operations of a transaction are completed successfully, the transaction is committed by the system, and all changes to the database are made permanent; the transaction cannot be rolled back once this is done.

2.0 Learning Outcomes

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

i. Know the meaning of transaction processing
ii. Describe the properties of transaction processing system
iii. State the features of transaction processing system
iv. Outline the benefits of transaction processing system

3.0 Learning Contents

3.1 Transaction Processing

A transaction is a logical unit of work (comprising one or more SQL statements) performed on the database to complete a common task and maintain data consistency. Transaction statements are closely related and perform interdependent actions. Each statement performs part of the task, but all of them are required for the complete task.

Transaction processing ensures that related data is added to or deleted from the database simultaneously, thus preserving data integrity in your application. In transaction processing, data is not written to the database until a commit command is issued. When this happens, data is permanently written to the database.

For example, if a transaction comprises database operations to update two database tables, either all updates are made to both tables, or no updates are made to either table. This condition guarantees that the data remains in a consistent state and the integrity of the data is maintained.

Transactions are used in a wide range of applications, such as banking systems, airline reservation systems, and organizational information systems. One of the primary goals of a transaction processing system is to allow several users to interact with the data in the system simultaneously while preserving the illusion that each user is executing alone

3.2 Transaction Properties

To ensure transactions are processed reliably, they should follow the ACID property. In reference to databases, ACID stands for Atomicity, Consistency, Isolation, and Durability.

Atomicity guarantees that all of the tasks of a transaction are performed or none of them are. For example, completing a holiday package from an online tour agency can fail for a multitude of reasons. There may be website or server problems, or the customer may not have enough credit in their bank account to pay for the whole package. Atomicity guarantees that one part of the package won't be debited if the others are not debited as well. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes

Consistency refers to the database being in a legal state when the transaction begins and when it ends. Consistency ensures a transaction can't break the rules or integrity constraints of the database. For example, suppose a specific transaction is intended to transfer money from the customer's bank account to that of a hotel owner and there is an error in the transaction logic. This error debits the customer's bank account but credits the airline instead of the hotel owner. The database is then said to be in an inconsistent state. Consistency ensures that any transaction in an inconsistent state will be aborted. Any data written to the database must be valid according to all defined rules, including but not limited to constraints, cascades, triggers, and any combination thereof.

Isolation refers to the ability of the application to make operations in a transaction appear isolated from all other operations. For example, a hotel company should be able to see specific hotel reservations made by the customer from the tour agency. However, they should not be able to see the customer's airline reservation or car rental details.

Durability refers to the guarantee that once the user has been notified of success, the transaction will not be undone. For example, if a tour package has been paid for and tickets printed out by the customer, a system failure from the agencies end should not affect the customer's holiday. Developers often ensure durability by having all transactions written to a log file that can be used to backup the system in case of failure. Hence, developers design systems to commit to a transaction only after it is safely in the log means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter).

Databases and transaction processing monitors ensure ACID properties through the use of locks, logs and two-phase commit as well as numerous other techniques.

Self Assessment Exercise(s)

1. What is the meaning of ACID?

Self Assessment Answer(s)

3.3 Benefits and Features of Transaction Processing

Transaction processing has the following benefits:

a. It allows sharing of computer resources among many users
b. It shifts the time of job processing to when the computing resources are less busy
c. It avoids idling the computing resources without minute-by-minute human interaction and supervision
d. It is used on expensive classes of computers to help amortize the cost by keeping high rates of utilization of those expensive resources

Features of Transaction Processing Systems

Rapid Processing

The rapid processing of transactions is vital to the success of any enterprise – now more than ever, in the face of advancing technology and customer demand for immediate action. TPS systems are designed to process transactions virtually instantly to ensure that customer data is available to the processes that require it.
Reliability
Similarly, customers will not tolerate mistakes. TPS systems must be designed to ensure that not only do transactions never slip past the net, but that the systems themselves remain operational permanently. TPS systems are therefore designed to incorporate comprehensive safeguards and disaster recovery systems. These measures keep the failure rate well within tolerance levels.

Standardisation
Transactions must be processed in the same way each time to maximise efficiency. To ensure this, TPS interfaces are designed to acquire identical data for each transaction, regardless of the customer.

Controlled Access

Since TPS systems can be such a powerful business tool, access must be restricted to only those employees who require their use. Restricted access to the system ensures that employees who lack the skills and ability to control it cannot influence the transaction process.

Self Assessment Exercise(s)

1. What are the benefits of transaction processing?

Self Assessment Answer(s)

Methodology

The basic principles of all transaction-processing systems are the same. However, the terminology may vary from one transaction-processing system to another, and the terms used below are not necessarily universal.

Commit

A commit is an explicit command to the database to permanently store the results of operations performed by a statement. This event successfully ends a transaction. In transaction processing, data is not written to the database until a commit command is issued. When this happens, data is permanently written to the database. You can use one of these ways to commit transactions: Auto commit or manual commit. An auto commit transaction writes database changes permanently immediately when the changes occur. A manual commit transaction will buffer database changes when they occur, and only write database changes permanently when the transaction is committed.

Rollback

A rollback is an explicit command to the database to cancel the results of operations performed by a statement. This event indicates that a transaction ended unsuccessfully.

Any failure to insert, update, or delete within a transaction boundary causes all record activity within that transaction to roll back. If no failures have occurred at the end of the transaction, a commit is done, and the records become available to other processes.

In the case of a catastrophic failure (such as due to network problems), the Database Management System (DBMS) performs an automatic rollback. Likewise, if the user clicks Cancel on a form, a rollback command can be issued through a system function.

Transaction-processing systems ensure database integrity by recording intermediate states of the database as it is modified, then using these records to restore the database to a known state if a transaction cannot be committed. For example, copies of information on the database *prior* to its modification by a transaction are set aside by the system before the transaction can make any modifications (this is sometimes called a *before image*). If any part of the transaction fails before it is committed, these copies are used to restore the database to the state it was in before the transaction began.

Rollforward

It is also possible to keep a separate journal of all modifications to a database (sometimes called *after images*). This is not required for rollback of failed transactions but it is useful for updating the database in the event of a database failure, so some transaction-processing systems provide it. If the database fails entirely, it must be restored from the most recent back-up. The back-up will not reflect transactions committed since the back-up was made. However, once the database is restored, the journal of after images can be applied to the database (*rollforward*) to bring the database up to date. Any transactions in progress at the time of the failure can then be rolled back. The result is a database in a consistent, known state that includes the results of all transactions committed up to the moment of failure.

Deadlocks

In some cases, two transactions may, in the course of their processing, attempt to access the same portion of a database at the same time, in a way that prevents them from proceeding. For example, transaction A may access portion X of the database, and transaction B may access portion Y of the database. If, at that point, transaction A then tries to access portion Y of the database while transaction B tries to access portion X, a *deadlock* occurs, and neither transaction can move forward. Transaction-processing systems are designed to detect these deadlocks when they occur.

Typically, both transactions will be cancelled and rolled back, and then they will be started again in a different order, automatically, so that the deadlock doesn't occur again. Or sometimes, just one of the deadlocked transactions will be cancelled, rolled back, and automatically re-started after a short delay.

Deadlocks can also occur between three or more transactions. The more transactions involved, the more difficult they are to detect, to the point that transaction processing systems find there is a practical limit to the deadlocks they can detect.

Compensating transaction

In systems where commit and rollback mechanisms are not available or undesirable, a compensating transaction is often used to undo failed transactions and restore the system to a previous state. This is often easily achieved using transactions and the commit/rollback mechanism Compensating transaction logic could be implemented as additional on top of database supporting commit/rollback. In that case we can decrease business transaction granularity.

Self Assessment Exercise(s)

1. Describe briefly rollback and commit mechanism?

Self Assessment Answer(s)

4.0 Conclusion

This unit discusses transaction processing which is a very important aspect of database management system,

5.0 Summary

You have learnt:

A transaction is any event that passes the ACID test in which data is generated or modified before storage in an information system

Atomicity: requires that each transaction is "all or nothing": if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged.

Consistency; to ensure that any transaction will bring the database from one valid state to another

Isolation: ensures that the concurrent execution of transactions results in a system state that could have been obtained if transactions are executed serially, i.e. one after the other.

Durability: means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter).

Features of transaction processing are Rapid Processing, Reliability, Standardisation and Controlled Access

6.0 Tutor-Marked Assignment

Discuss the features of transaction processing systems

7.0 References/ Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill, Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://en.wikipedia.org/wiki/ACID

http://www.hypergurl.com/blog/databases/

http://docs.oracle.com/cd/E17984_01/doc.898/e14706/transaction_processing. htm

http://www.bestpricecomputers.co.uk/glossary/transaction-processing-systems.htm

http://my.safaribooksonline.com/book/databases/9781558606234/two-phase-commit/ch08

Unit 2

Overview of Distributed Databases

Contents

1.0 Introduction

A distributed database is a database in which storage devices are not all attached to a common CPU. This means relevant information may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. The computers will however be located in the same physical location or they will be linked through a network. Distributed databases thus make collection of data easily distributable across multiple locations and even thought this may seem little impressing one must thing about the advantages. This type of database software makes it possible for different individuals to use data stored on another computer. In an office for instance, this database implies that the same information must not be stored on each computer because the different users may access it directly from a different computer. These databases are in the end meant to improve the performance of the end-user and they work as such.

2.0 Learning Outcomes

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

i. State the meaning of distributed database
ii. Explain the basis of distributed database
iii. Explain the types of distributed database
iv. Explain distributed data storage
v. Discuss the concept of replication

3.0 Learning Contents

3.1 Distributed Databases Definition

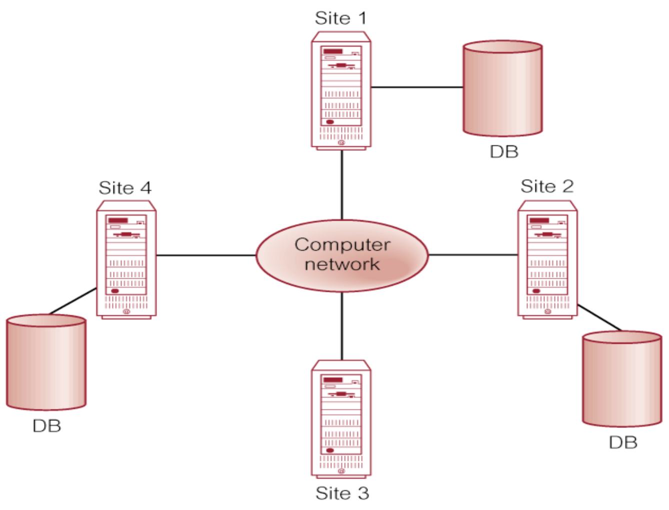
In a **distributed database system**, data is physically stored across several sites, and each site is typically managed by a DBMS that is capable of running independently of the other sites. The location of data items and the degree of autonomy of individual sites have a significant impact on all aspects of the system, including query optimization and processing, concurrency control, and recovery. In contrast to parallel databases, the distribution of data is governed by factors such as local ownership and increased availability, in addition to performance issues

A **parallel database system on** the other hand is one that seeks to improve performance through parallel implementation of various operations such as loading data, building indexes, and evaluating queries. Although data may be stored in a distributed fashion in such a system, the distribution is governed solely by performance considerations.

Unlike parallel systems, in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely coupled sites that share no physical components

A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. The replication and distribution of databases improves database performance at end-user worksites

Figure 1: A distributed database



Distributed Properties

Data in a distributed database system is stored across several sites, and each site is typically managed by a DBMS that can run independently of the other sites. The classical view of a distributed database system is that the system should make the impact of data distribution **transparent**.

In particular, the following properties are considered desirable:

Distributed data independence: Users should be able to ask queries without specifying where the referenced relations, or copies or fragments of the relations, are located. This principle is a natural extension of physical and logical data independence. Further, queries that span multiple sites should be optimized systematically in a cost-based manner, taking into account communication costs and difeerences in local computation costs.

Distributed transaction atomicity: Users should be able to write transactions that access and update data at several sites just as they would write transactions over purely local data. In particular, the effects of a transaction across sites should continue to be atomic; that is, all changes persist if the transaction commits, and none persist if it aborts

Self Assessment Exercise(s)

1. Distinguish between Parallel and Distributed Database?

Self Assessment Answer(s)

3.2 Advantages and Disadvantages of distributed Database

Advantages of distributed database include:

i. There is easy access to the information. If one terminal stops working, the replication and distribution process makes sure that the same information can be accessed by another terminal.
ii. Different data can have different transparency levels, meaning user privileges can be assigned to maintain a hierarchy of information sharing.
iii. Distributed database system is cheaper than other network server setups. The individual units do not need to have advanced processing power and simply act as workstations on the computer network.
iv. There is Local autonomy which means that users at one site have control over their data
v. It is reliable/available which means that in case of site crash or communication failure part of the database will be available.
vi. It gives room for improved performance due to proximate of data to the users
vii. There is room for expandability/ scalability which makes the system to be easily expanded.

Disadvantages of distributed database are

i. Complexity of management and control. Applications must recognize data location and they must be able to stitch together data from various sites.
ii. Technologically difficult: Data integrity, transaction management, concurrency control, security, backup, recovery, query optimization, access path selection are all issues that must be addressed.
iii. Security lapses have increased instances when data are in multiple locations.
iv. Lack of standards due to the absence of communication protocols can make the processing and distribution of data difficult.
v. Increased storage and infrastructure requirements because multiple copies of data are required at various separate locations which would require more disk space.
vi. Increased costs due to the higher complexity of training.
vii. Requires duplicate infrastructure (personnel, software and licensing, physical location/environment) and these can sometimes offset any operational savings.
viii. Threat of Security: Control of network Security and data control

3.3 Distributed Data Storage

Consider a relation *r* that is to be stored in the database. There are two approaches to storing this relation in the distributed database:

i. **Replication**. The system maintains several identical replicas (copies) of the relation, and stores each replica at a different site. The alternative to replication is to store only one copy of relation *r*.
ii. **Fragmentation**. The system partitions the relation into several fragments, and stores each fragment at a different site. Fragmentation and replication can be combined: A relation can be partitioned into several fragments and there may be several replicas of each fragment. In the following subsections, we elaborate on each of these techniques.

Data Replication

If relation *r* is replicated, a copy of relation *r* is stored in two or more sites. In the most extreme case, we have **full replication**, in which a copy is stored in every site in the system.

There are a number of advantages and disadvantages to replication.

Increased Availability of Data. If one of the sites containing relation *r* fails, then the relation *r*can be found in another site. Similarly, if local copies of remote relations are available, we are less vulnerable to failure of communication links Thus, the system can continue to process queries involving *r*, despite the failure of one site.

Increased parallelism. In the case where the majority of accesses to the relation*r* result in only the reading of the relation, then several sites can process queries involving *r* in parallel. The more replicas of *r* there are, the greater the chance that the needed data will be found in the site where the transactions executing. Hence, data replication minimizes movement of data between sites.

Faster query evaluation: Queries can execute faster by using a local copy of a relation instead of going to a remote site.

Increased overhead on update. The system must ensure that all replicas of a relation *r* are consistent; otherwise, erroneous computations may result. Thus, whenever *r* is updated, the update must be propagated to all sites containing replicas. The result is increased overhead. For example, in a banking system, where account information is replicated in various sites, it is necessary to ensure that the balance in a particular account agrees in all sites.

Replication can be synchronous or asynchronous

Data Fragmentation

If relation *r* is fragmented, *r* is divided into a number of *fragments* *r*1, *r*2*, . . . , rn*. These fragments contain sufficient information to allow reconstruction of the original relation *r*. There are two different schemes for fragmenting a relation: *horizontal* fragmentation and *vertical* fragmentation. Horizontal fragmentation splits the relation by assigning each tuple of *r* to one or more fragments. Vertical fragmentation splits the relation by decomposing the scheme *R* of relation *r*.

We shall illustrate these approaches by fragmenting the relation *account*, with the schema

Account-schema = (*account-number*, *branch-name*, *balance*)

In **horizontal fragmentation**, a relation *r* is partitioned into a number of subsets, r_1, $r_2, \ldots, r_n$. Each tuple of relation *r* must belong to at least one of the fragments, so that the original relation can be reconstructed, if needed.

As an illustration, the *account* relation can be divided into several different fragments, each of which consists of tuples of accounts belonging to a particular branch. If the banking system has only two branches—Hillside and Valleyview—then there are two different fragments:

$account_1 = \sigma_{branch\text{-}name = \text{“Hillside”}}(account)$

$account_2 = \sigma_{branch\text{-}name = \text{“Valleyview”}}(account)$

Horizontal fragmentation is usually used to keep tuples at the sites where they are used the most, to minimize data transfer.

In general, a horizontal fragment can be defined as a *selection* on the global relation *r*. That is,we use a predicate P_i to construct fragment r_i:

$r_i = \sigma_{P_i}(r)$

We reconstruct the relation *r* by taking the union of all fragments; that is,

$r = r_1 \cup r_2 \cup \cdots \cup r_n$

In our example, the fragments are disjoint. By changing the selection predicates used to construct the fragments, we can have a particular tuple of *r* appear in more than one of the r_i.

In its simplest form, vertical fragmentation is the same as decomposition.

Vertical fragmentation of *r*(*R*) involves the definition of several subsets of attributes $R_1, R_2, \ldots, R_n$ of the schema *R* so that$R = R_1 \cup R_2 \cup \cdots \cup R_n$

Each fragment r_i of *r* is defined by

$r_i = \Pi_{R_i}(r)$

The fragmentation should be done in such a way that we can reconstruct relation *r*

from the fragments by taking the natural join

$r = r_1 \bowtie r_2 \bowtie r_3 \bowtie \cdots \bowtie r_n$

One way of ensuring that the relation *r* can be reconstructed is to include the primary-key attributes of *R* in each of the *Ri*. More generally, any superkey can be used. It is often convenient to add a special attribute, called a *tuple-id*, to the schema *R*. The tuple-id value of a tuple is a unique value that distinguishes the tuple from all other tuples. The tuple-id attribute thus serves as a candidate key for the augmented schema, and is included in each of the *Ri*s. The physical or logical address for a tuple can be used as a tuple-id, since each tuple has a unique address.

To illustrate vertical fragmentation, consider a university database with a relation *employee-info* that stores, for each employee, *employee-id*, *name*, *designation*, and *salary*.

For privacy reasons, this relation may be fragmented into a relation *employee-privateinfo* containing *employee-id* and *salary*, and another relation *employee-public-info* containing attributes *employee-id*, *name*, and *designation*. These may be stored at different sites, again for security reasons.

The two types of fragmentation can be applied to a single schema; for instance, the fragments obtained by horizontally fragmenting a relation can be further partitioned vertically. Fragments can also be replicated. In general, a fragment can be replicated, replicas of fragments can be fragmented further, and so on.

Self Assessment Exercise(s)

1. What are the two approaches in distributed used in database storage?

Self Assessment Answer(s)

3.4 Distributed Database Transparency Features

In a distributed system the users and administrators should with proper implementation, interact with the system as if the system was centralized. This transparency allows for the functionality desired in such a structured system without special programming requirements, allowing for any number of local and/or remote tables to be accessed at a given time across the network.

The different types of transparency sought after in a DDBMS are data distribution transparency, heterogeneity transparency, transaction transparency, and performance transparency

1. **Data transparency.** Requires that the user of a distributed database system should not be required to know either where the data are physically located or how the data can be accessed at the specific local site. It can take the following forms:
    a. Fragmentation transparency. Users are not required to know how a relation has been fragmented.

b. Replication transparency. Users view each data object as logically unique. The distributed system may replicate an object to increase either system performance or data availability. Users do not have to be concerned with what data objects have been replicated, or where replicas have been placed.

c. Location transparency. Users are not required to know the physical location of the data. The distributed database system should be able to find any data as long as the data identifier is supplied by the user transaction.

2. **Heterogeneity Transparency** requires that the user should not be aware of the fact that they are using a different DBMS if they access data from a remote site. The user should be able to use the same language that they would normally use at their regular access point and the DDBMS should handle query language translation if needed.
3. **Transaction Transparency** requires that the DDBMS guarantee that concurrent transactions do not interfere with each other (concurrency transparency) and that it must also handle database recovery (recovery transparency).
4. **Performance Transparency** mandates that the DDBMS should have a comparable level of performance to a centralized DBMS. Query optimizers can be used to speed up response time.

System Failure Modes

A distributed system may suffer from the same types of failure that a centralized system does (for example, software errors, hardware errors, or disk crashes). There are, however, additional types of failure with which we need to deal in a distributed environment. The basic failure types are

i. Failure of a site
ii. Loss of messages
iii. Failure of a communication link
iv. Network partition

The loss or corruption of messages is always a possibility in a distributed system. The system uses transmission-control protocols, such as TCP/IP, to handle such errors. Information about such protocols may be found in standard textbooks on networking

However, if two sites *A* and *B* are not directly connected, messages from one to the other must be *routed* through a sequence of communication links. If a communication

link fails, messages that would have been transmitted across the link must be rerouted. In some cases, it is possible to find another route through the network, so that the messages are able to reach their destination. In other cases, a failure may result in there being no connection between some pairs of sites. A system is **partitioned** if it has been split into two (or more) subsystems, called **partitions**, that lack any connection between them. Note that, under this definition, a subsystem may consist of a single node.

Self Assessment Exercise(s)

1. Mention the types of transparency in distributed database?

Self Assessment Answer(s)

3.5 Types of Distributed Databases

Homogenous Distributed Database: This involves that data is distributed but all servers run the same DBMS software. All sites have identical database management system software, are aware of one another, and agree to cooperate in processing users' requests. In such a system, local sites surrender a portion of their autonomy in terms of their right to change schemas or database management system software. That software must also cooperate with other sites in exchanging information about transactions, to make transaction processing possible across multiple sites.

Heterogenous Disrtibuted Database: This involves different sites run under the control of different DBMSs, essentially autonomously, and are connected somehow to enable access to data from multiple sites. It is also referred to as a multidatabase system. Different sites may use different schemas, and different database management system software. The sites may not be aware of one another, and they may provide only limited facilities for cooperation in transaction processing. The differences in schemas are often a major problem for query processing, while the divergence in software becomes a hindrance for processing transactions that access multiple sites.

3.6 Distributed DBMS Architectures

There are three alternative approaches to separating functionality across different DBMS-related processes; these alternative distributed DBMS architectures are called *Client-Server*, *Collaborating Server*, and *Middleware*.

Client-Server Systems

A **Client-Server** system has one or more client processes and one or more server processes, and a client process can send a query to any one server process. Clients are responsible for user-interface issues, and servers manage data and execute transactions.

Thus, a client process could run on a personal computer and send queries to a server running on a mainframe. This architecture has become very popular for several reasons. First, it is relatively simple to implement due to its clean separation of functionality and because the server is centralized. Second, expensive server machines are not underutilized by dealing with mundane user-interactions, which are now relegated to inexpensive client machines.

Third, users can run a graphical user interface that they are familiar with, rather than the (possibly unfamiliar and unfriendly) user interface on the server. While writing Client-Server applications, it is important to remember the boundary between the client and the server and to keep the communication between them as set-oriented as possible. In particular, opening a cursor and fetching tuples one at a time generates many messages and should be avoided. (Even if we fetch several tuples and cache them at the client, messages must be exchanged when the cursor is advanced to ensure that the current row is locked.) Techniques to exploit client-side caching to reduce communication overhead have been extensively studied.

Collaborating Server Systems

The Client-Server architecture does not allow a single query to span multiple servers because the client process would have to be capable of breaking such a query into appropriate subqueries to be executed at di_erent sites and then piecing together the answers to the subqueries. The client process would thus be quite complex, and its capabilities would begin to overlap with the server; distinguishing between clients and servers becomes harder. Eliminating this distinction leads us to an alternative to the Client-Server architecture: a **Collaborating Server** system. We can have a collection of database servers, each capable of running transactions against local data, which cooperatively execute transactions spanning multiple servers. When a server receives a query that requires access to data at other servers, it generates appropriate subqueries to be executed by other servers and puts the results together to compute answers to the original query. Ideally, the decomposition of the query should be done using cost-based optimization, taking into account the costs of network communication as well as local processing costs.

Middleware Systems

The Middleware architecture is designed to allow a single query to span multiple servers, without requiring all database servers to be capable of managing such multisite execution strategies. It is especially attractive when trying to integrate several legacy systems, whose basic capabilities cannot be extended. The idea is that we need just one database server that is capable of managing queries and transactions spanning multiple servers; the remaining servers only need to handle local queries and transactions. We can think of this special server as a layer of software that coordinates the execution of queries and transactions across one or more independent database servers; such software is often called **middleware**. The middleware layer is capable of executing joins and other relational operations on data obtained from the other servers, but typically, does not itself maintain any data.

Self Assessment Exercise(s)

1. Name the three (3) Distributed Databases Architecture?

Self Assessment Answer(s)

4.0 Conclusion

This unit discusses a brief overview of the concept of distributed database system with discussion on features, types, architectures, advantages and disadvantages.

5.0 Summary

You have learnt:

1. A distributed database system consists of a collection of sites, each of which maintains a local database system. Each site is able to process local transactions: those transactions that access data in only that single site
2. Distributed databases may be homogeneous, where all sites have a common schema and database system code, or heterogeneous, where the schemas and system codes may differ.
3. There are several issues involved in storing a relation in the distributed database, including replication and fragmentation. It is essential that the system minimize the degree to which a user needs to be aware of how a relation is stored.
4. A distributed system may suffer from the same types of failure that can afflict a centralized system. There are, however, additional failures with which we need to deal in a distributed environment, including the failure of a site, the failure of a link, loss of a message, and network partition. Each of these problems needs to be considered in the design of a distributed recovery scheme.
5. **Data transparency.** Requires that the user of a distributed database system should not be required to know either where the data are physically located or how the data can be accessed at the specific local site. It can take the following forms: It can be Fragmentation transparency, Replication transparency, Location transparency and
6. **Heterogeneity Transparency**: requires that the user should not be aware of the fact that they are using a different DBMS if they access data from a remote site.
7. **Transaction Transparency** requires that the DDBMS guarantee that concurrent transactions do not interfere with each other (concurrency transparency) and that it must also handle database recovery (recovery transparency).
8. **Performance Transparency** mandates that the DDBMS should have a comparable level of performance to a centralized DBMS. Query optimizers can be used to speed up response time.

6.0 Tutor-Marked Assignment

1. What are the advantages and disadvantages of distributed database system?
2. Under which situations will it be beneficial to have replication or fragmentation of data?

7.0 References/Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://ogsa-dai.sourceforge.net/documentation/ogsadai4.0/ogsadai4.0-gt/DQPOverview.html

http://docs.oracle.com/cd/B10501_01/server.920/a96521/ds_txns.htm

http://docs.oracle.com/cd/B10500_01/server.920/a96521/ds_concepts.htm#15768

http://www.csc.liv.ac.uk/~dirk/Comp332/COMP332-DDB-notes.pdf

http://it.toolbox.com/blogs/enterprise-solutions/overview-of-distributed-databases-16228

http://databasemanagement.wikia.com/wiki/Distributed_Database_Management_System_(DDBMS)

Unit 3

Physical Database Design

Contents

1.0 Introduction

Physical database design is concerned with transforming the logical database structures into an internal model consisting of stored records, files, indexes and other physical structures. Databases are used to store information in the form of files of records and are typically stored on magnetic disks. This unit focuses on the file Organisation in DBMS, the access methods available and the system parameters associated with them. File Organisation is the way the files are arranged on the disk and access method is how the data can be retrieved based on the file Organisation.

2.0 Learning Outcomes

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

i. Define storage of databases on hard disks;
ii. Explain the various storage devices used in storing files
iii. Discuss the various file Organisation techniques;
iv. Describes database tuning

3.0 Learning Contents

3.1 Physical database Design

The purpose of physical database design is to translate the logical description of data into the technical specifications for storing and retrieving data for the DBMS. The goal is to create a design for storing data that will provide adequate performance and ensure database integrity, security and recoverability. More so, The goal of a database system is to simplify and facilitate access to data
Physical File is a file as stored on the disk. The main issues relating to physical files are:

i. Constructs to link two pieces of data: i.e Sequential storage and Pointers.
ii. File Organisation: How the files are arranged on the disk?
iii. Access Method: How the data can be retrieved based on the file Organisation

3.2 Overview of Physical Storage Media

Storage media are classified by speed of access, cost per unit of data to buy the media, and by the medium's reliability. Unfortunately, as speed and cost go up, the reliability does down. Among the media typically available are these:

a) **Cache** is the fastest and the most costly for of storage. The type of cache referred to here is the type that is typically built into the CPU chip and is 256KB, 512KB, or 1MB. Thus, cache is used by the operating system and has no application to database, per se.

b) **Main memory** is the volatile memory in the computer system that is used to hold programs and data. While prices have been dropping at a staggering rate, the increases in the demand for memory have been increasing faster. Today's 32-bit computers have a limitation of 4GB of memory. This may not be sufficient to hold the entire database and all the associated programs, but the more memory available will increase the response time of the DBMS. There are attempts underway to create a system with the most memory that is cost effective, and to reduce the functionality of the operating system so that only the DBMS is supported, so that system response can be increased. However, the contents of main memory are lost if a power failure or system crash occurs.

c) **Flash memory** is also referred to as *electrically erasable programmable read-only memory (EPROM)*. flash memory differs from main memory in that data survive power failure. Reading data from flash memory takes less than 100 nanoseconds (a nanosecond is 1/1000 of a microsecond), which is roughly as fast as reading data from main memory. However, writing data to flash memory is more complicated—data can be written once, which takes about 4 to 10 microseconds, but cannot be overwritten directly. To overwrite memory that has been written already, we have to erase an entire bank of memory at once; it is then ready to be written again. A drawback of flash memory is that it can support only a limited number of erase cycles, ranging from 10,000 to 1 million. Flash memory has found popularity as a replacement for magnetic disks for storing small volumes of data (5 to 10 megabytes) in low-cost computer systems, such as computer systems that are embedded in other devices, in hand-held computers, and in other digital electronic devices such as digital cameras. Since it is small (5 to 10MB) and expensive, it has little or no application to the DBMS.

d) **Magnetic-disk storage** is the primary medium for long-term on-line storage today. Prices have been dropping significantly with a corresponding increase in capacity. New disks today are in excess of 20GB. Unfortunately, the demands have been increasing and the volume of data has been increasing faster. The organizations using a DBMS are always trying to keep up with the demand for storage. This media is the most cost-effective for on-line storage for large databases.

e) **Optical storage** is very popular, especially CD-ROM systems. This is limited to data that is read-only. It can be reproduced at a very low-cost and it is expected to grow in popularity, especially for replacing written manuals.

The most popular forms of optical storage are the *compactdisk* (CD), which can hold about 640 megabytes of data, and the *digital video disk* (DVD) which can hold 4.7 or 8.5 gigabytes of data per side of the disk (or up to 17 gigabytes on a two-sided disk). Data are stored optically on a disk, and are read by a laser. The optical disks used in read-only compact disks (CD-ROM) or read-only digital video disk (DVD-ROM) cannot be written, but are supplied with data prerecorded.

There are "record-once" versions of compact disk (called CD-R) and digital video disk (called DVD-R), which can be written only once; such disks are also called **write-once, read-many** (WORM) disks. There are also "multiple-write" versions of compact disk (called CD-RW) and digital video disk (DVD-RW and DVD-RAM), which can be written multiple times. Recordable compact disks are magnetic–optical storage devices that use optical means to read magnetically encoded data. Such disks are useful for archival storage of data as well as distribution of data. Jukebox systems contain a few drives and numerous disks that can be loaded into one of the drives automatically (by a robot arm) on demand.

(f) Tape storage is used for backup and archival data. It is cheaper and slower than all of the other forms, but it does have the feature that there is no limit on the amount of data that can be stored, since more tapes can be purchased. As the tapes get increased capacity, however, restoration of data takes longer and longer, especially when only a small amount of data is to be restored. This is because the retrieval is sequential, the slowest possible method.

Tapes have a high capacity (40 gigabyte to 300 gigabytes tapes are currently available), and can be removed from the tape drive, so they are well suited to cheap archival storage. Tape jukeboxes are used to hold exceptionally large collections of data, such as remote-sensing data from satellites, which could include as much as hundreds of terabytes (1 terabyte = 1012 bytes), or even a petabyte (1 petabyte = 1015 bytes) of data.

Self Assessment Exercise(s)

1. How can you classify storage media?

Self Assessment Answer(s)

3.3 File Organisation and Its Types

A file organisation is a technique to organise data in the secondary memory. It deals with obtaining data representation for files on external storage devices so that required functions (e.g. retrieval, update) may be carried out efficiently.

File Organisation is a way of arranging the records in a file when the file is stored on the disk. Data files are organized so as to facilitate access to records and to ensure their efficient storage. A tradeoff between these two requirements generally exists: if rapid access is required, more storage is required to make it possible. Selection of File Organisations is dependant on two factors as shown below:

a) Typical DBMS applications need a small subset of the DB at any given time.
b) When a portion of the data is needed it must be located on disk, copied to memory for processing and rewritten to disk if the data was modified.

A file of record is likely to be accessed and modified in a variety of ways, and different ways of arranging the records enable different operations over the file to be carried out efficiently. A DBMS supports several file Organisation techniques. The important task of the DBA is to choose a good Organisation for each file, based on its type of use.

The particular organisation most suitable for any application will depend upon such factors as the kind of external storage available, types of queries allowed, number of keys, mode of retrieval and mode of update.

Heap files (unordered file)

Basically these files are unordered files. It is the simplest and most basic type. These files consist of randomly ordered records. The records will have no particular order. The operations we can perform on the records are insert, retrieve and delete. The features of the heap file or the pile file Organisation are:

i. New records can be inserted in any empty space that can accommodate them.
ii. When old records are deleted, the occupied space becomes empty and available for any new insertion.
iii. If updated records grow; they may need to be relocated (moved) to a new empty space. This needs to keep a list of empty space.

Advantages of heap files

1. This is a simple file Organisation method.
2. Insertion is somehow efficient.
3. Good for bulk-loading data into a table.
4. Best if file scans are common or insertions are frequent.

Disadvantages of heap files

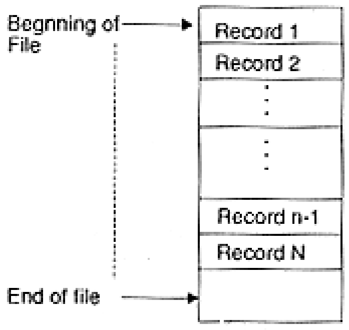
1. Retrieval requires a linear search and is inefficient.
2. Deletion can result in unused space/need for reorganisation.

Sequential File Organisation

The most basic way to organise the collection of records in a file is to use sequential Organisation. Records of the file are stored in sequence by the primary key field values. They are accessible only in the order stored, i.e., in the primary key order. This kind of file Organisation works well for tasks which need to access nearly every record in a file, e.g., payroll. Let us see the advantages and disadvantages of it.
In a sequentially organised file records are written consecutively when the file is created and must be accessed consecutively when the file is later used for input A sequential file maintains the records in the logical sequence of its primary key values. Sequential files are inefficient for random access, however, are suitable for sequential access. A sequential file can be stored on devices like magnetic tape that allow sequential access.
On an average, to search a record in a sequential file would require to look into half of the records of the file. However, if a sequential file is stored on a disk (remember disks support direct access of its blocks) with keyword stored separately from the rest of record, then only those disk blocks need to be read that contains the desired reorder records. This type of storage allows binary search on sequential file blocks, thus, enhancing the speed of access.

Updating a sequential file usually creates a new file so that the record sequence on primary key is maintained. The update operation first copies the records till the record after which update is required into the new file and then the updated record is put followed by the remainder of records. Thus method of updating a sequential file automatically creates a backup copy.
Additions in the sequential files are also handled in a similar manner to update. Adding a record requires shifting of all records from the point of insertion to the end of file to create space for the new record. On the other hand deletion of a record requires a compression of the file space.
The basic advantages of sequential file is the sequential processing, as next record is easily accessible despite the absence of any data structure. However, simple queries are time consuming for large files. A single update is expensive as new file must be created, therefore, to reduce the cost per update, all updates requests are sorted in the order of the sequential file. This update file is then used to update the sequential file in a single go. The file containing the updates is sometimes referred to as a transaction file.
This process is called the batch mode of updating. In this mode each record of master sequential file is checked for one or more possible updates by comparing with the update information of transaction file. The records are written to new master file in the sequential manner. A record that require multiple update is written only when all the updates have been performed on the record. A record that is to be deleted is not written to new master file. Thus, a new updated master file will be created from the transaction file and old master file.
Thus, update, insertion and deletion of records in a sequential file require a new file creation. Can we reduce creation of this new file? Yes, it can easily be done if the original sequential file is created with holes which are empty records spaces. Thus, a reorganisation can be restricted to only a block that can be done very easily within the main memory. Thus, holes increase the performance of sequential file insertion and deletion. This organisation also support a concept of overflow area, which can store the spilled over records if a block is full. This technique is also used in index sequential file organisation.
The figure below shows this organisation

Figure1: sequential file organisation



Advantages of Sequential File Organisation

i. It is fast and efficient when dealing with large volumes of data that need to be processed periodically (batch system).

Disadvantages of sequential File Organisation

i. Requires that all new transactions be sorted into the proper sequence for sequential access processing.
ii. Locating, storing, modifying, deleting, or adding records in the file require rearranging the file.
iii. This method is too slow to handle applications requiring immediate updating or responses.

Indexed (Indexed Sequential) File Organisation

It organises the file like a large dictionary, i.e., records are stored in order of the key but an index is kept which also permits a type of direct access. The records are stored sequentially by primary key values and there is an index built over the primary key field.

The retrieval of a record from a sequential file, on average, requires access to half the records in the file, making such inquiries not only inefficient but very time consuming for large files. To improve the query response time of a sequential file, a type of indexing technique can be added.
An index is a set of index value, address pairs. Indexing associates a set of objects to a set of orderable quantities, that are usually smaller in number or their properties. Thus, an index is a mechanism for faster search. Although the indices and the data blocks are kept together physically, they are logically distinct. Let us use the term an index file to describes the indexes and let us refer to data files as data records. An index can be small enough to be read into the main memory.

A sequential (or sorted on primary keys) file that is indexed on its primary key is called an index sequential file. The index allows for random access to records, while the sequential storage of the records of the file provides easy access to the sequential records. An additional feature of this file system is the over flow area. The overflow area provides additional space for record addition without the need to create.

Hashed File Organisation

Hashing is the most common form of purely random access to a file or database. It is also used to access columns that do not have an index as an optimisation technique. Hash functions calculate the address of the page in which the record is to be stored based on one or more fields in the record. The records in a hash file appear randomly distributed across the available space. It requires some hashing algorithm and the technique. Hashing Algorithm converts a primary key value into a record address. The most popular form of hashing is division hashing with chained overflow.

Advantages of Hashed file Organisation

i. Insertion or search on hash-key is fast.
ii. Best if equality search is needed on hash-key.

Disadvantages of Hashed file Organisation

i. It is a complex file Organisation method.
ii. Search is slow.
iii. It suffers from disk space overhead.
iv. Unbalanced buckets degrade performance.
v. Range search is slow.

B-tree:

B-tree is a tree data structure that keeps data sorted and allows searches, insertions, and deletions in logarithmic amortized time. It is most commonly used in databases and filesystems. It is a dynamic, multilevel index, with maximum and minimum bounds on the number of keys in each index segment (usually called a "block" or "node"). In a B+ tree, in contrast to a B-tree, all records are stored at the leaf level of the tree; only keys are stored in interior nodes.

In B-trees, internal (non-leaf) nodes can have a variable number of child nodes within some pre-defined range. When data is inserted or removed from a node, its number of child nodes changes. In order to maintain the pre-defined range, internal nodes may be

joined or split. Because a range of child nodes is permitted, B-trees do not need rebalancing as frequently as other self-balancing search trees, but may waste some space, since nodes are not entirely full. The lower and upper bounds on the number of child nodes are typically fixed for a particular implementation. For example, in a 2-3 B-tree(often simply referred to as a 2-3 tree), each internal node may have only 2 or 3 child nodes

Self Assessment Exercise(s)

1. What are the various file organization method?

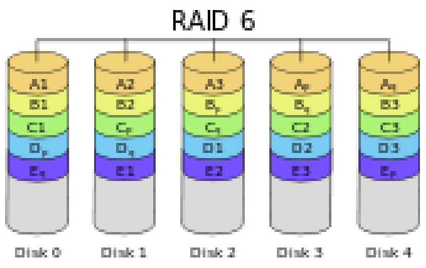
Self Assessment Answer(s)

3.4 Database Tuning

Database tuning is comprised of a group of activities used to optimize and regulate the performance of a database. It refers to configuration of the database files, the database management system (DBMS), as well as the hardware and operating system on which the database is hosted. The goal of database tuning is to maximize the application of system resources in an attempt to execute transactions as efficiently and quickly as possible. The large majority of DBMS are designed with efficiency in mind; however, it is possible to enhance a database's performance via custom settings and configurations.

The tuning of a database management system centers around the configuration of memory and the processing resources of the computer running the DBMS. This can involve setting the recovery interval of the DMBS, establishing the level of concurrency control, and assigning which network protocols are used to communicate throughout the database. Memory utilized by the DBMS is allocated for data, execution procedures, procedure cache, and work space. Since it is faster to directly access data in memory than data on storage, it is possible to decrease the average access time of database transactions by maintaining a decent sized data cache. Database performance can also be improved by using the cache to store execution procedures as they would not need to be recompiled with every transaction. By assigning processing resources to specific functions and activities, it is also possible to improve the concurrency of the system. “Database concurrency controls ensure that transactions occur in an ordered fashion. The main job of these controls is to protect transactions issued by different users/applications from the effects of each other. They must preserve the four characteristics of database transactions: atomicity, isolation, consistency and durability” (About.com).

Input/Output(I/O) tuning is another major component of database tuning. I/O tuning mainly deals with database transaction logs. Database transaction logs are files that are associated with temporary work spaces as well as both table and index file storage. Transaction logs and temporary spaces are heavy consumers of I/O, and affect performance for all users of the database. Placing them appropriately is crucial. The main goal of I/O tuning a database is to optimize and balance the read and write transactions of the system in order to achieve an increased speed in database transactions and a decreased database access time. Another method of ensuring that a database is fast and reliable is the Use of RAID in the creation of



the database. RAID stands for Redundant Array of Independent Disks. Here is an example as to why RAID is superior to a single disk. If data are stored on one disk, the entire database is completely reliant on that one disk; if it were to fail, the database would not exist anymore. Another drawback to having it on a single disk is the read/write time. One hard disk can only be so fast. If there is a lot of I/O data being processed, it can be a lengthy process. One thing that RAID does is it divides and replicates the data onto several independent disks. Instead of having all our eggs in one basket, we have diversified our risk or disk failure away. If we had a RAID 6 array with 4 drives, the tolerance for failure is 2 disks. This means that if 2 hard drives fail completely, the database will still function perfectly. On top of failure tolerance, another great upside to using RAID is tasks are performed faster. There is an increase in speed equal to this multiplication factor: (n−2)X. Reading is faster, and writing is faster because instead of one disk trying to find all the data, the task is broken down into parts, and each hard disk does part of the job.

Another common part of database tuning revolves around database maintenance. Database maintenance includes things such as backing up the database as well as the defragmentation of the data residing within the database. When a database is under heavy use, transaction log entries must be removed in order to create space for future entries. Since regular transaction log backups are smaller in size, they take less time to complete and, therefore, interrupt scheduled database activity for much shorter periods of time. Much like a computer system, the defragmentation of database tables and indexes greatly increase the efficiency of the database's ability in accessing data. A database's level of fragmentation depends on the data's nature, how the data is manipulated, and the amount of free space left in database pages.

Self Assessment Exercise(s)

1. What do your understand by RAID?

Self Assessment Answer(s)

4.0 Conclusion

This unit focuses on physical database design and discusses various storage media, file organization and data tuning.

5.0 Summary

You have learnt that:

1. Physical database design is concerned with transforming the logical database structures into an internal model consisting of stored records, files, indexes and other physical structures
2. Storage media are classified by speed of access, cost per unit of data to buy the media, and by the medium's reliability
3. Example of storage media are cache, main memory, flash memory, tape and optical disk

4. File Organisation is a way of arranging the records in a file when the file is stored on the disk
5. Typical file organization include heap, sequential , hashed and indexed sequential
6. B-tree is a tree data structure that keeps data sorted and allows searches, insertions, and deletions in logarithmic amortized time
7. Database tuning is comprised of a group of activities used to optimize and regulate the performance of a database. It refers to configuration of the database files, the database management system (DBMS), as well as the hardware and operating system on which the database is hosted

6.0 Tutor-Marked Assignment

1. Classify file organization. Why reorganization is required in sequential file organization
2. What are the physical characteristics of magnetic disks?
3. Describe the storage structure of indexed sequential files

7.0 References/Further Reading

Atzeni P., Cori S., Paraboschi S., Tortone R., (1999) Database System: Concept, System and Architecture; McGrawHill

Fred R. M., Jeffrey A.H., Mary B. P.(2005). Modern Database Management (7th ed.), Prentice Hall

Ramakrishnan, R. and Gehrke, J.,(2003) *Database Management Systems,* 3rd ed., McGraw-Hill, New York.

Raghu R. & Johannes G. (2002) Database Management Systems (3rd ed.). McGraw-Hill,

Silberschatz K.S., (2004). Database system concept (4th Ed.). McGraw Hill Company

http://www.smckearney.com/hncdb/notes/lec.physicaldesign.2up.pdf

http://db.grussell.org/section014.html

http://webserver.ignou.ac.in/virtualcampus/BIT_StudyMaterials/tri8/cst203-bl1-u3.htm

http://searchstorage.techtarget.com/definition/RAID

http://www.inf.unibz.it/dis/teaching/DMS/ln/dms02.pdf

http://databasemanagement.wikia.com/wiki/Database_Tuning

http://www.cecs.csulb.edu/~monge/classes/share/B+TreeIndexes.html

http://vedyadhara.ignou.ac.in/wiki/images/f/ff/MCS-023-_Unit_-4(File_Organisation_in_DBMS).pdf

ANSWERS TO SELF ASSESSMENT QUESTIONS

MODULE 1
UNIT 1

SAQ 1: Distinguish between data, information, field and record

SAQ2 : Peter Chen
SQL become standard language IN1980

SAQ3: 1. Banking 2. Airlines /Railways/Road Transport 3. Universities 4. Credit Card Transaction 5. Telecommunication 6. Finance 7. Sales 8. On-line Retailers 9. Manufacturing 10. Human Resources 11. Internet

SAQ4: ADVANTAGES OF A DBMS

Data independence, efficient data access, Data integrity and security,
Data administration, Concurrent access and crash recovery, Reduced application development time:

***Disadvantages of a DBMS:*Cost, Security, Large size Complexity, Greater impact of failure, More difficult recovery**

UNIT2

SAQ 1 Hierarchy, Network, Relational, Object and Object Relational model
SAQ 2: Homogenous and heterogeneous
Single user and multi user system

UNIT 3

Users, hardware, software and data

Naive, sophisticated and specialized

UNIT 4

SAQ1: It provides data independence

Conceptual level

Physical level

UNIT :

1 A strong entity set has a primary key. All tuples in the set are distinguishable by that key. A weak entity set has no primary key unless attributes of the strong entity set on which it depends are included. Tuples in a weak entity set are partitioned according to their relationship with tuples in a strong entity set. Tuples

within each partition are distinguishable by a discriminator, which is a set of attributes.

2. Constraints used in E-R model: 1. Cardinality Constraints 2. Participation Constraints 3. Key Constraints

3 Relationship

4 Cardinality

UNIT 2

SAQ1: Tuple and attribute

SAQ 2The basic difference between E-R diagram and schema diagram is that E-R diagrams do not show foreign key attributes explicitly, whereas schema diagrams show them explicitly.

UNIT 3

SAQ 1: Elimination of redundant data storage. Close modeling of real world entities, processes, and their relationships. Structuring of data so that the model is flexible.

update anomalies, addition anomalies, and deletion anomalies. All these can easily be avoided through data normalization. Data redundancy produces data integrity problems, caused by the fact that data entry failed to conform to the rule that all copies of redundant data must be identical.

MODULE 3

UNIT 1

SAQ1 CREATE, ALTER, DROP, USE
SAQ 2 Insert, select, delete update

UNIT 2

SAQ Πename (σcname = "First Bank Corporation" (works))
Πename, city (σcname = "First Bank Corporation" (employee ⋈works))
Π ename (employee ⋈works ⋈company)

UNIT 3

SAQ (i) Tuple Calculus:

{t[Emp#] | t Î ASSIGNED_TO Ù "p (p Î PROJECT _ $u (u Î ASSIGNED_TO Ù

p[Project#] = u[Project#] Ù t[Emp#] = u[Emp#]))}

Domain Calculus:

{e | $p (<p, e> Î ASSIGNED_TO Ù "p1 (<p1, n1, c1> Î PROJECT

_ <p1, e> Î ASSIGNED_TO))}

(ii) Tuple Calculus:

{t[Emp#] | t Î ASSIGNED_TO Ù ¬$u (u Î ASSIGNED_TO

Ù u[Project#] = 'COMP123'Ù t[Emp#] = u[Emp#])}

Domain Calculus:

{e | $p (<p, e> Î ASSIGNED_TO Ù "p1, e1 (<p1, e1> Ï ASSIGNED_TO

Ú p1 ¹ 'COMP123' Ú e1 ¹ e))}

2 a. Find the id of all students who took the same course in two different semesters.

{t.SID| Student(t)AND9t1 2 Takes, 9t2 2 Takes(t1.SID = t.SIDANDt2.SID = t.SID AND t1.CourseID = t2.CourseID AND t1.Semester <> t2.Semester)}

b. Find the name of all students who never got an F.

{t.SName | Student(t)ANDnot(9t1 2 Takes(t1.SID = t.SIDANDt1.Grade =0 F0))}

or by carrying the "Not" inside, we get the expression:

{t.SName | Student(t)AND8t1 2 Takes(t1.SID <> t.SIDORt1.Grade <>0 F0)}

UNIT 4

SAQ 1: SELECT, UPDATE, INSERT,DELETE

SAQ2: security, interactive DML

SAQ3: mysql, mysql server, oracle and ms access

UNIT 5

SAQ1 Ans: (i) SELECT NAME FROM STUDENT WHERE MAJOR = 'CS'

(ii) SELECT COURSE_NAME FROM COURSE C, SECTION S

WHERE C.COURSE# = S.COURSE#

  AND INSTRUCTOR = 'JOHN' AND YEAR =2012

OR

```
SELECT COURSE_NAME FROM COURSE
WHERE COURSE# IN (SELECT COURSE# FROM SECTION
WHERE INSTRUCTOR = ‘JOHN’ AND YEAR = 2012)
```

(iii) DELETE FROM STUDENT WHERE NAME = ‘Smith’ AND STUDENT# = 17

(iv) INSERT INTO COURSE
VALUES(‘Knowledge Engineering’, ‘CS4390’, 3, ‘CS’)

MODULE 4
UNIT 1

SAQ1 ACID means Atomicity, consistency, Integrity and Durability

1. Benefits includes

a. It allows sharing of computer resources among many users

b. It shifts the time of job processing to when the computing resources are less busy

c. It avoids idling the computing resources without minute-by-minute human interaction and supervision

d. It is used on expensive classes of computers to help amortize the cost by keeping high rates of utilization of those expensive resources

2. Rollback is a database concept that where a transaction fails or is aborted and its actions are undone in order to restore the database to its previous consistent state.
A commit is an explicit command to the database to permanently store the results of operations performed by a statement

UNIT 2
SAQ1:In a distributed database system, data is physically stored across several sites, and each site is typically managed by a DBMS that is capable of running independently of the other sites. The location of data items and the degree of autonomy of individual sites have a significant impact on all aspects of the system, including query optimization and processing, concurrency control, and recovery.

While A parallel database system on the other hand is one that seeks to improve performance through parallel implementation of various operations such as loading data, building indexes, and evaluating queries. Although data may be stored in a distributed fashion in such a system, the distribution is governed solely by performance considerations.

SAQ2: Replication and fragmentation
SQQ3: data, heterogeneity, transaction and performance
SAQ4:*Client-Server*, *Collaborating Server*, and *Middleware*.

UNIT 3
SAQ1: they can be classified by cost, speed and medium reliability

SAQ2: Sequential, hash, indexed sequential and heaped

SAQ3: RAID (*Redundant Array of Independent Disks*) is a technology which makes use of two or more hard drives in order to improve performance, reliability or create larger data volumes