NATIONAL OPEN UNIVERSITY OF NIGERIA

COURSE CODE :MBA 858

COURSE TITLE:
DATABASE MANAGEMENT APPLICATION SYSTEM

COURSE GUIDE

MBA 858
DATABASE MANAGEMENT APPLICATION SYSTEM

Course Writer Gerald C. Okereke
Eco Communications Inc.
Lagos Ikeja

Course Editor Mr. E. Eseyin
National Open University of Nigeria

Programme Leader Dr. O. J. Onwe
National Open University of Nigeria

Course Coordinator Bimbola, E.U. Adegbola
National Open University of Nigeria

NATIONAL OPEN UNIVERSITY OF NIGERIA

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

Abuja Office
5, Dar es Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja
Nigeria

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

Published by
National Open University of Nigeria

Printed 2009

ISBN: 978-058-331-9

CONTENTS **PAGE**

Introduction

This course, Database Management System (DBMS), is a course designed in the pursuit of a degree in Masters Degrees in business, finance, marketing and related fields of study. It is also a course that can be studied by Postgraduate Diploma students in business, sciences and education.

This course is relevant to students studying business because information/data form the foundation of any business enterprise. Thus a thorough understanding of how to manipulate, design and manage databases.

This course is primarily to be studied by students who are already graduates or post graduates in any field of study. Students who had not had exposure to computer science in their first degrees need to put in extra effort to grasp this course properly.

This course guide takes you through the nature of the course, the materials you are going to use and how you are to use materials to your maximum benefit. It is expected that at least two hours should be devoted to the study of each course unit. For each unit there assessments in the form of tutor-marked assignment. You are advised carry out the exercises immediately after studying the unit.

There will be tutorial lectures to organized for this course. This serves as an avenue to interact with course instructors who will communicate more clearly with you regarding the course. You are advised to attend the tutorial lectures because it will enhance your understanding of the course. Note that it is also through these tutorial lectures that you will submit your tutor-marked assignment and be assessed accordingly.

Course Aim

Behind the development and design of this course is to know how to design, manipulate and manage databases. The course participants are exposed to the various forms, types and models of database systems to enable them make viable choices. Supportive and complimentary concepts of managing data and documents are thoroughly examined to give a wholesome view of data/information management. The ultimate aim is to encourage the usage of database management systems for effective data management.

Course Objectives

The following are the major objectives of this course:

- define a DATABASE MANAGEMENT APPLICATION SYSTEM
- give a description of the Database Management structure
- define a Database
- define basic foundational terms of Database
- understand the applications of Databases
- know the advantages and disadvantages of the different models
- compare relational model with the Structured Query Language (SQL)
- know the constraints and controversies associated with relational database model.
- know the rules guiding transaction ACID
- identify the major types of relational management systems
- compare and contrast the types of RDBMS based on several criteria
- understand the concept of data planning and Database design
- know the steps in the development of Databases
- trace the history and development process of SQL
- know the scope and extension of SQL
- differentiate Discretionary and. Mandatory Access Control Policies
- know the Proposed OODBMS Security Models
- identify the various functions of Database Administrator
- trace the history and development process of datawarehouse
- list various benefits of datawarehouse
- compare and contrast document management system and content management systems
- know the basic components of document management systems

Course Materials

1. Course Guide
2. Study Units
3. Textbooks
4. Assignment File
5. Tutorials

Study Units

This course consists of thirteen (13) units, divided into 3 modules. Each module deals with major aspect of the course.

Module 1

Module 2

Module 3

In studying the units, a minimum of 2 hours is expected of you. Start by going through the unit objectives for you to know what you need to learn and know in the course of studying the unit. At the end of the study of the unit, evaluate yourself to know if you have achieved the objectives of the unit. If not, you need to go through the unit again.

To help you ascertain how well you understood the course, there will be exercises mainly in the form of tutor-marked assignments at the end of each unit. At first attempt, try to answer the questions without necessarily having to go through the unit. However, if you cannot proffer solutions offhand, then go through the unit to answer the questions.

Assignment File

For each unit, you will find one (1) or two (2) tutor-marked assignments. These assignments serve two purposes:

1. **Self Evaluation: The tutor-marked assignment will assists** you to thoroughly go through each unit, because you are advised to attempt to answer the questions immediately after studying each unit. The questions are designed in such a way that at least one question must prompt a typical self assessment test.

2. Obtain Valuable Marks: The tutor-marked assignment is also a valid means to obtain marks that will form part of your total score in this course. It constitutes 30% of total marks obtainable in this course.

You are advised to go through the units thoroughly or you to be able to proffer correct solution to the tutor-marked assignment

Assessment

You will be assessed and graded in this course through tutor-marked assignment and formal written examination. The allocation of marks is as indicated below.

• Assignments = 30 %
• Examination = 70%

Final examination and grading

The final examination will consist of two (2) sections:

1. Section 1: This is compulsory and weighs 40 marks
2. Section 2: This consists of six (6) questions out of which you are to answer (4) questions. It weights 60 marks.

The duration of the examination will be 3 hours.

Credit Units

This course attracts 3 credit units only.

Presentation Schedule

This constitutes the scheduled dates and venue for tutorial classes, as well as how and when to submit the tutorials. All this will be communicated to you in due course.

Course Overview

This indicates the units/topic, issues to be studied each week. It also includes the duration of the course, revision week and examination week. The details are as provided below:

Unit	Title of Work	Week's Activity	Assessment (end of unit)
	Course Guide		
	Module 1		
1	Overview	1	TMA
2	Database	2	TMA
3	Database Concepts	3	TMA
4	Database Models 1	4	TMA
5	Database Models: Relational Model	5	TMA
6	Basic Components of DBMS	6	TMA
	Module 2		
1	Development and Design-Of Database	7	TMA
2	Structured Query Languages (SQL)	8	TMA
3	Database and Information Systems Security	9	TMA
4	Database Administrator and Administration	10	TMA
	Module 3		
1	Relational Database Management Systems	11	TMA
2	Datawarehouse	12	TMA
3	Document Management System	13	TMA
	Revision and Examination	14	

Course Code MBA 858

Course Title DATABASE MANAGEMENT APPLICATION SYSTEM

Course Writer Gerald C. Okereke
Eco Communications Inc.
Lagos Ikeja

Course Editor Mr. E. Eseyin
National Open University of Nigeria

Programme Leader Dr. O. J. Onwe
National Open University of Nigeria

Course Coordinator Bimbola, E.U. Adegbola
National Open University of Nigeria

NATIONAL OPEN UNIVERSITY OF NIGERIA

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

Abuja Office
5, Dar es Salaam Street
Off Aminu Kano Crescent
Wuse II, Abuja
Nigeria

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

Published by
National Open University of Nigeria

Printed 2009

ISBN: 978-058-331-9

CONTENTS PAGE

MODULE 1

UNIT 1 OVERVIEW

CONTENTS

1.0 INTRODUCTION

A Database Management System (DBMS) is computer software designed for the purpose of managing databases based on a variety of data models.

2.0 OBJECTIVES

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

- define a DATABASE MANAGEMENT APPLICATION SYSTEM
- give a description of the Database Management Structure
- numerate the benefits of DATABASE MANAGEMENT APPLICATION SYSTEM
- describe the features and capabilities of a typical DBMS
- identify and differentiate the different types and models of DBMS.

3.0 MAIN CONTENT

3.1 Description

A DBMS is a complex set of software programs that controls the organization, storage, management, and retrieval of data in a database. DBMS are categorized according to their data structures or types; sometime DBMS is also known as Data base Manager. It is a set of prewritten programs that are used to store, update and retrieve a Database. A DBMS includes:

A modeling language to define the schema of each database hosted in the DBMS, according to the DBMS data model.

•The four most common types of organizations are the hierarchical, network, relational and object models. Inverted lists and other methods are also used. A given DATABASE MANAGEMENT APPLICATION SYSTEM may provide one or more of the four models. The optimal structure depends on the natural organization of the application's data, and on the application's requirements (which include transaction rate (speed), reliability, maintainability, scalability, and cost).

•The dominant model in use today is the ad hoc one embedded in SQL, despite the objections of purists who believe this model is a corruption of the relational model, since it violates several of its fundamental principles for the sake of practicality and performance. Many DBMSs also support the Open Database Connectivity API that supports a standard way for programmers to access the DBMS.

Data structures (fields, records, files and objects) optimized to deal with very large amounts of data stored on a permanent data storage device (which implies relatively slow access compared to volatile main memory).

A database query language and report writer to allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data.

•It also controls the security of the database.

•Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called subschemas. For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data.

•If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However, it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.

A transaction mechanism, that ideally would guarantee the ACID properties, in order to ensure data integrity, despite concurrent user accesses (concurrency control), and faults (fault tolerance).

•It also maintains the integrity of the data in the database.

•The DBMS can maintain the integrity of the database by not allowing more than one user to update the same record at the same time. The DBMS can help prevent duplicate records via unique index constraints; for example, no two customers with the same customer numbers (key fields) can be entered into the database.

•The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data.

When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system.

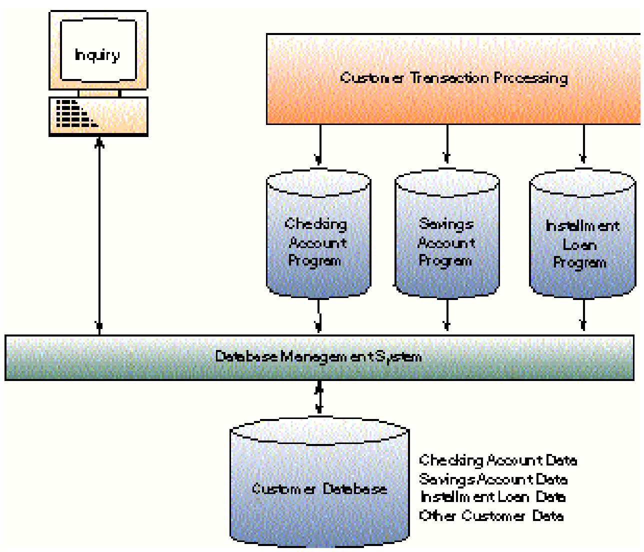
Organizations may use one kind of DBMS for daily transaction processing and then move the detail onto another computer that uses another DBMS better suited for random inquiries and analysis. Overall systems design decisions are performed by data administrators and systems analysts. Detailed database design is performed by database administrators.

Database servers are specially designed computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers, with RAID disk arrays used for stable storage. Connected to one or more servers via a high-speed channel, hardware database accelerators are also used in large volume transaction processing environments.
DBMSs are found at the heart of most database applications. Sometimes DBMSs are built around a private multitasking kernel with built-in networking support although nowadays these functions are left to the operating system.

3.2 DBMS Benefits

- •Improved strategic use of corporate data
- •Reduced complexity of the organization's information systems environment
- •Reduced data redundancy and inconsistency
- •Enhanced data integrity
- •Application-data independence
- •Improved security
- •Reduced application development and maintenance costs
- •Improved flexibility of information systems
- •Increased access and availability of data and information
- •Logical & Physical data independence
- •Concurrent access anomalies.
- •Facilitate atomicity problem.
- •Provides central control on the system through DBA.

***Figure 1: An example of a database management approach in** a banking information system.*



Note how the savings, checking, and installment loan programs use a database management system to share a customer database. Note also that the DBMS allows a user to make a direct, ad hoc interrogation of the database without using application programs.

3.3 Features and Capabilities of DBMS

A DBMS can be characterized as an "attribute management system" where attributes are small chunks of information that describe something. For example, "colour" is an attribute of a car. The value of the attribute may be a color such as "red", "blue" or "silver".

Alternatively, and especially in connection with the relational model of database management, the relation between attributes drawn from a specified set of domains can be seen as being primary. For instance, the database might indicate that a car that was originally "red" might fade to "pink" in time, provided it was of some particular "make" with an inferior paint job. Such higher arity relationships provide information on all of the underlying domains at the same time, with none of them being privileged above the others.

Throughout recent history specialized databases have existed for scientific, geospatial, imaging, and document storage and like uses. Functionality drawn from such applications has lately begun appearing in mainstream DBMSs as well. However, the main focus there, at least when aimed at the commercial data processing market, is still on descriptive attributes on repetitive record structures.

Thus, the DBMSs of today roll together frequently-needed services or features of attribute management. By externalizing such functionality to the DBMS, applications effectively share code with each other and are relieved of much internal complexity. Features commonly offered by DATABASE MANAGEMENT APPLICATION SYSTEMs include:

Query Ability

Querying is the process of requesting attribute information from various perspectives and combinations of factors. Example: "How many 2-door cars in Texas are green?"

A database query language and report writer allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data. It also controls the security of the database.

Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called subschemas. For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data.

If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.

Backup and Replication

Copies of attributes need to be made regularly in case primary disks or other equipment fails. A periodic copy of attributes may also be created for a distant organization that cannot readily access the original. DBMS usually provide utilities to facilitate the process of extracting and disseminating attribute sets.

When data is replicated between database servers, so that the information remains consistent throughout the database system and users cannot tell or even know which server in the DBMS they are using, the system is said to exhibit replication transparency.

Rule Enforcement

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

Security

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

Computation

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

Change and Access Logging

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

Automated Optimization

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

3.4 Uses Of DATABASE MANAGEMENT APPLICATION SYSTEMs

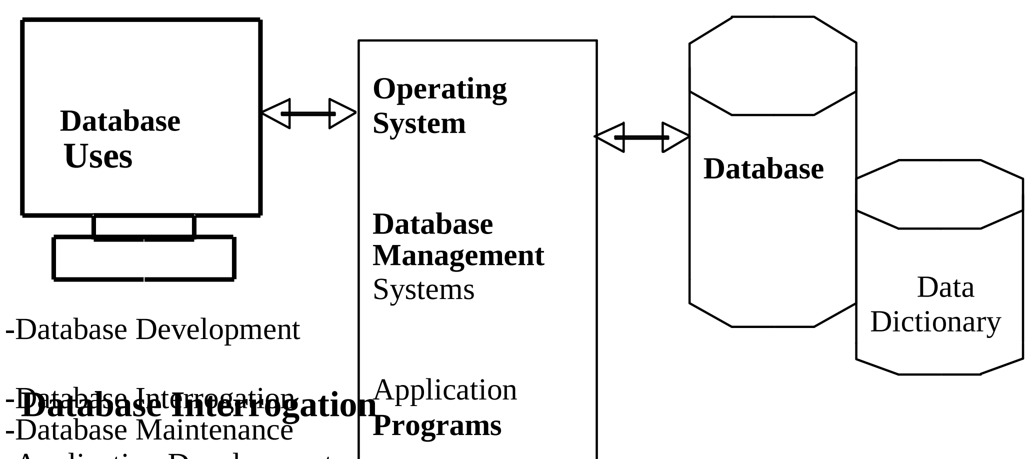
The four major uses of DATABASE MANAGEMENT APPLICATION SYSTEMs are:

1. Database Development
2. Database Interrogation
3. Database Maintenance
4. Application Development

Database Development

Database packages like Microsoft Access, Lotus Approach allow end users to develop the database they need. However, large organizations with client/server or mainframe-based system usually place control of enterprise-wide database development in the hands of database administrators and other database specialists. This improves the integrity and security of organizational database. Database developers use the data definition languages (DDL) in database management systems like oracle 9i or IBM's BD2 to develop and specify the data contents, relationships and structure each databases, and to modify these database specifications called a data dictionary.

Figure 2: The Four Major Uses of DBMS



Database Interrogation

The Database interrogation capability is a major use of Database management system. End users can interrogate a database management system by asking for information from a database using a query *language or a report generator. They can receive an immediate* response in the form of video displays or printed reports. No difficult programming ideas are required.

Database Maintenance

The databases of organizations need to be updated continually to reflect new business transactions and other events. Other miscellaneous changes must also be made to ensure accuracy of the data in the database. This database maintenance process is accomplished by transaction processing programs and other end-user application packages within the support of the DATABASE MANAGEMENT APPLICATION SYSTEM. End-users and information specialists can also employ various utilities provided by a DBMS for database maintenance.

Application Development

DATABASE MANAGEMENT APPLICATION SYSTEM packages play major roles in application development. End-users, systems analysts and other application developers can use the fourth generational languages (4GL) programming languages and built-in software development tools provided by many DBMS packages to develop custom application programs. For example you can use a DBMS to easily develop the data entry screens, forms, reports, or web pages by a business application. A database management system also makes the job of application programmers easier, since they do not have to develop detailed data handling procedures using a conventional programming language every time they write a program.

3.5 Models

The various models of DATABASE MANAGEMENT APPLICATION SYSTEMs are:

1. Hierarchical
2. Network
3. Object-oriented
4. Associative
5. Column-Oriented
6. Navigational
7. Distributed
8. Real Time Relational
9. SQL

These models will be discussed in details in subsequent units of this course.

3.6 List of DATABASE MANAGEMENT APPLICATION SYSTEMs Software

Examples of DBMSs include

- Oracle
- DB2
- Sybase Adaptive Server Enterprise
- FileMaker
- Firebird
- Ingres
- Informix
- Microsoft Access
- Microsoft SQL Server
- Microsoft Visual FoxPro
- MySQL
- PostgreSQL
- Progress
- SQLite
- Teradata
- CSQL
- OpenLink Virtuoso

4.0 CONCLUSION

DATABASE MANAGEMENT APPLICATION SYSTEMs has continue to make data management and storage to be much easier than it used to be. With the emergence of relational model of database management systems much of the big challenge in handling large database has been reduced. More database management products will be available on the market as there will be improvement in the already existing once.

5.0 SUMMARY

- A Database Management System (DBMS) is computer software designed for the purpose of managing databases based on a variety of data models.
- A DBMS is a complex set of software programs that controls the organization, storage, management, and retrieval of data in a database
- When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system.
- Often it is desirable to limit who can see or change which attributes or groups of attributes. This may be managed directly by individual, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groups to roles which are then granted entitlements.

- A DBMS can be characterized as an "attribute management system" where attributes are small chunks of information that describe something. For example, "colour" is an attribute of a car. The value of the attribute may be a color such as "red", "blue" or "silver".
- Querying is the process of requesting attribute information from various perspectives and combinations of factors. Example: "How many 2-door cars in Texas are green?"
- As computers grew in capability, this trade-off became increasingly unnecessary and a number of general-purpose database systems emerged; by the mid-1960s there were a number of such systems in commercial use. Interest in a standard began to grow, and Charles Bachman, author of one such product, IDS, founded the Database *Task Group within CODASYL*

6.0 TUTOR-MARKED ASSIGNMENT

1. Mention 10 DATABASE MANAGEMENT APPLICATION SYSTEMs software
2. Describe briefly the backup and replication ability of database management systems.

7.0 REFERENCES/FURTHER READINGS

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

O'Brien, James A. 2003, Introduction to Information Systems, McGraw-Hill, 11th Edition

UNIT 2 DATABASE

CONTENTS

1.0 INTRODUCTION

A Database is a structured collection of data that is managed to meet the needs of a community of users. The structure is achieved by organizing the data according to a database model. The model in most common use today is the relational model. Other models such as the hierarchical model and the network model use a more explicit representation of relationships (see below for explanation of the various database models).

A computer database relies upon software to organize the storage of data. This software is known as a database management system (DBMS). Databases management systems are categorized according to the database model that they support. The model tends to determine the query languages that are available to access the database. A great deal of the internal engineering of a DBMS, however, is independent of the data model, and is concerned with managing factors such as performance, concurrency, integrity, and recovery from hardware failures. In these areas there are large differences between products.

2.0 OBJECTIVES

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

- define a database
- define basic foundational terms of database
- know a little bit of the history of the development of database
- know and differentiate the different types of database
- answer the question of the structure of database.

3.0 MAIN CONTENT

3.1 Foundations of Database Terms

File

A file is an ordered arrangement of records in which each record is stored in a unique identifiable location. The sequence of the record is then the means by which the record will be located. In most computer systems, the sequence of records is either alphabetic or numeric based on field common to all records such as name or number.

Records

A record or tuple is a complete set of related fields. For example, the *Table 1 below shows a set of related fields, which is a record. In other* words, if this were to be a part of a table then we would call it a row of data. Therefore, a row of data is also a record.

Table 1

Sr No	Icode	Ord No	Ord Date	Pqty
1	RKSK-T	0083/99	3/3/2008	120

Field

A field is a property or a characteristic that holds some piece of information about an entity. Also, it is a category of information within a set of records. For example, the first names, or address or phone numbers of people listed in address book.

Relations

In the relational data model, the data in a database is organized in relations. A relation is synonymous with a'table'. A table consists of columns and rows, which are referred as field and records in DBMS terms, and attributes and tuples in Relational DBMS terms.

Attributes

An attribute is a property or characteristics that hold some information about an entity. A 'Customer' for example, has attributes such as a name, and an address.

Table 2: DBMS and Relational DBMS Terms in Comparison

Common Term	DBMS Terminology	RDBMS Terminology
Database Table		Database
Table	Table	Relation
Column Field		Attribute
Row	Record Tuple	

3.2 History

The earliest known use of the term database was in November 1963, when the System Development Corporation sponsored a symposium under the title Development and Management of a Computer-centered *Data Base. Database as a single word became common in Europe in the* early 1970s and by the end of the decade it was being used in major American newspapers. (The abbreviation DB, however, survives.)

The first DATABASE MANAGEMENT APPLICATION SYSTEMs were developed in the 1960s. A pioneer in the field was Charles Bachman. Bachman's early papers show that his aim was to make more effective use of the new direct access storage devices becoming available: until then, data processing had been based on punched cards and magnetic tape, so that serial processing was the dominant activity. Two key data models arose at this time: CODASYL developed the network model based on Bachman's ideas, and (apparently independently) the hierarchical model was used in a system developed by North American Rockwell later adopted by IBM as the cornerstone of their IMS product. While IMS along with the CODASYL IDMS were the big, high visibility databases developed in the 1960s, several others were also born in that decade, some of which have a significant installed base today. The relational model was proposed by E. F. Codd in 1970. He criticized existing models for confusing the abstract description of information structure with descriptions of physical access mechanisms. For a long while, however, the relational model remained of academic interest only. While CODASYL products (IDMS) and network model products (IMS) were conceived as practical engineering solutions taking account of the technology as it existed at the time, the relational model took a much more theoretical perspective, arguing (correctly) that hardware and software technology would catch up in time. Among the first implementations were Michael Stonebraker's Ingres at Berkeley, and the System R project at IBM. Both of these were research prototypes, announced during 1976. The first commercial products, Oracle and DB2, did not appear until around 1980.

During the 1980s, research activity focused on distributed database systems and database machines. Another important theoretical idea was the Functional Data Model, but apart from some specialized applications in genetics, molecular biology, and fraud investigation, the world took little notice.

In the 1990s, attention shifted to object-oriented databases. These had some success in fields where it was necessary to handle more complex data than relational systems could easily cope with, such as spatial databases, engineering data (including software repositories), and multimedia data.

In the 2000s, the fashionable area for innovation is the XML database. As with object databases, this has spawned a new collection of start-up companies, but at the same time the key ideas are being integrated into the established relational products.

3.3 Database Types

Considering development in information technology and business applications, these have resulted in the evolution of several major types of databases. Figure 1 illustrates several major conceptual categories of databases that may be found in many organizations.

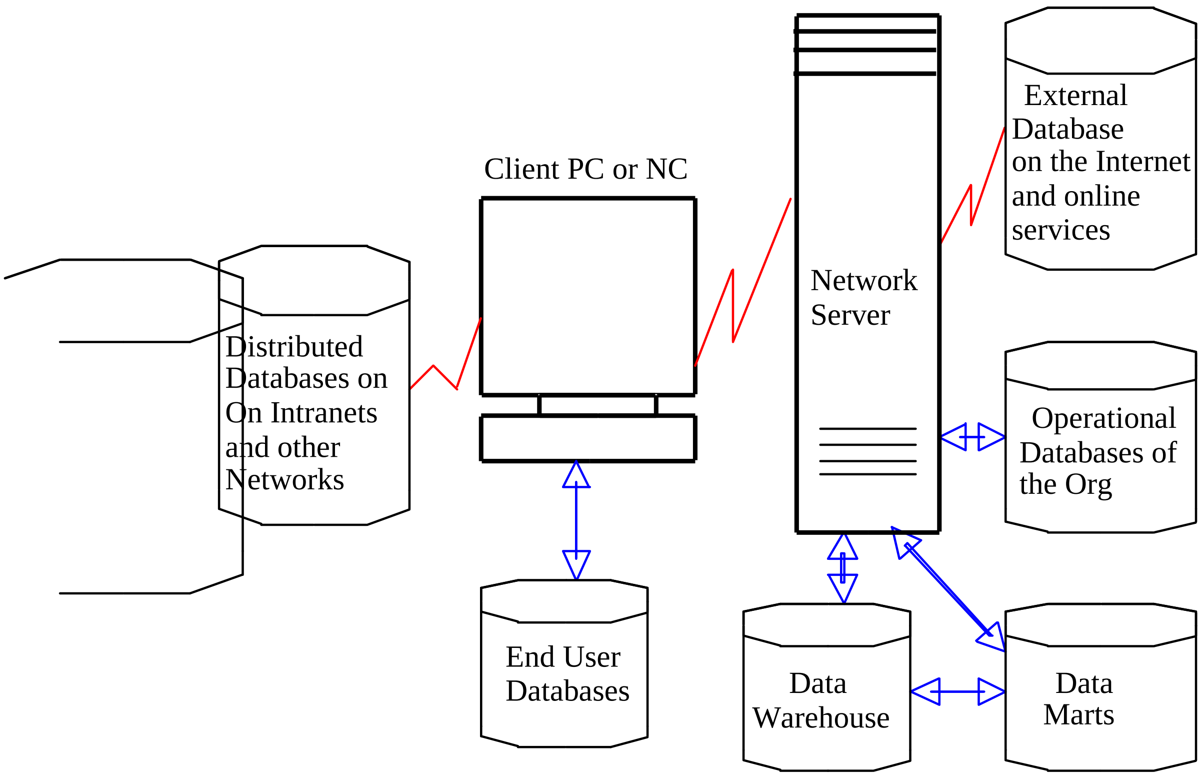
Operational Database

These databases store detailed data needed to support the business processes and operations of the e-business enterprise. They are also called subject area databases (SDDB), transaction database and *production databases. Examples are a customer database, human* resources databases, inventory databases, and other databases containing data generated by business operations. This includes databases on Internet and e-commerce activity such as click stream data, describing the online behaviour of customers or visitors to a company website.

Distributed Databases

Many organizations replicate and distribute copies or parts of databases to network servers at a variety of sites. They can also reside in network servers at a variety of sites. These distributed databases can reside on network servers on the World Wide Web, on corporate intranets or extranets or on any other company networks. Distributed databases may be copies of operational or analytic databases, hypermedia or discussion databases, or any other type of database. Replication and distribution of databases is done to improve database performance and security. Ensuring that all of the data in an organization’s distributed databases are consistently and currently updated is a major challenge of distributed database management.

Figure 1: Examples of the major types of databases used by organizations and end users.



External Databases

Access to wealth of information from external databases is available for a fee from conventional online services, and with or without charges from many sources on the Internet, especially the world wide web. Websites provide an endless variety of hyperlinked pages of multimedia documents in hypermedia databases for you to access. Data are available in the form of statistics in economics and demographic activity from statistical data banks. Or you can view or download abstracts or complete copies of newspapers, magazines, newsletters, research papers, and other published materials and other periodicals from bibliographic and full text databases.

3.4 Database Storage Structures

Database tables/indexes are typically stored in memory or on hard disk in one of many forms, ordered/unordered Flat files, ISAM, Heaps, Hash buckets or B+ Trees. These have various advantages and disadvantages discussed in this topic. The most commonly used are B+trees and ISAM.

Methods

Flat Files

A flat file database describes any of various means to encode a data model (most commonly a table) as a plain text file.

A flat file is a file that contains records, and in which each record is specified in a single line. Fields from each record may simply have a fixed width with padding, or may be delimited by whitespace, tabs, commas (CSV) or other characters. Extra formatting may be needed to avoid delimiter collision. There are no structural relationships. The data are "flat" as in a sheet of paper, in contrast to more complex models such as a relational database.

The classic example of a flat file database is a basic name-and-address list, where the database consists of a small, fixed number of fields: *Name, Address, and Phone Number. Another example is a simple* HTML table, consisting of rows and columns. This type of database is routinely encountered, although often not expressly recognized as a database.

Implementation: It is possible to write out by hand, on a sheet of paper, a list of names, addresses, and phone numbers; this is a flat file database. This can also be done with any typewriter or word processor. But many pieces of computer software are designed to implement flat file databases.

Unordered storage typically stores the records in the order they are inserted, while having good insertion efficiency, it may seem that it would have inefficient retrieval times, but this is usually never the case as most databases use indexes on the primary keys, resulting in efficient retrieval times.

Ordered or Linked list storage typically stores the records in order and may have to rearrange or increase the file size in the case a record is inserted, this is very inefficient. However is better for retrieval as the records are pre-sorted (Complexity O(log(n))).

Structured files

• simplest and most basic method

- insert efficient, records added at end of file – ‘chronological’ order
- retrieval inefficient as searching has to be linear
- deletion – deleted records marked

- requires periodic reorganization if file is very volatile

• advantages

- good for bulk loading data
- good for relatively small relations as indexing overheads are avoided
- good when retrievals involve large proportion of records

• disadvantages

- not efficient for selective retrieval using key values, especially if large
- sorting may be time-consuming

• not suitable for 'volatile' tables

Hash Buckets

• 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

- Hashing functions chosen to ensure that addresses are spread evenly across the address space
- 'occupancy' is generally 40% – 60% of total file size
- unique address not guaranteed so collision detection and collision resolution mechanisms are required

• open addressing
• chained/unchained overflow
• pros and cons

- efficient for exact matches on key field
- not suitable for range retrieval, which requires sequential storage
- calculates where the record is stored based on fields in the record
- hash functions ensure even spread of data
- collisions are possible, so collision detection and restoration is required

B+ Trees

These are the most used in practice.

• the time taken to access any tuple is the same because same number of nodes searched
• index is a full index so data file does not have to be ordered
• Pros and cons

- versatile data structure – sequential as well as random access
- access is fast
- supports exact, range, part key and pattern matches efficiently
- ‘volatile’ files are handled efficiently because index is dynamic – expands and contracts as table grows and shrinks

Less well suited to relatively stable files – in this case, ISAM is more efficient.

3.5 Database Servers

A database server is a computer program that provides database services to other computer programs or computers, as defined by the client-server model. The term may also refer to a computer dedicated to running such a program. Database management systems frequently provide database server functionality, and some DBMS's (e.g., MySQL) rely exclusively on the client-server model for database access.

In a master-slave model, database master servers are central and primary locations of data while database slave servers are synchronized backups of the master acting as proxies.

3.6 Database Replication

Database replication can be used on many database management systems, usually with a master/slave relationship between the original and the copies. The master logs the updates, which then ripple through to the slaves. The slave outputs a message stating that it has received the update successfully, thus allowing the sending (and potentially re-sending until successfully applied) of subsequent updates.

Multi-master replication, where updates can be submitted to any database node, and then ripple through to other servers, is often desired, but introduces substantially increased costs and complexity which may make it impractical in some situations. The most common challenge that exists in multi-master replication is transactional conflict prevention or resolution. Most synchronous or eager replication solutions do conflict prevention, while asynchronous solutions have to do conflict resolution. For instance, if a record is changed on two nodes simultaneously, an eager replication system would detect the conflict before confirming the commit and abort one of the transactions. A lazy replication system would allow both transactions to commit and run a conflict resolution during resynchronization.

Database replication becomes difficult when it scales up. Usually, the scale up goes with two dimensions, horizontal and vertical: horizontal scale up has more data replicas, vertical scale up has data replicas located further away in distance. Problems raised by horizontal scale up can be alleviated by a multi-layer multi-view access protocol. Vertical scale up runs into less trouble when the Internet reliability and performance are improving.

3.7 Relational Database

A relational database is a database that conforms to the relational model, and refers to a database's data and schema (the database's structure of how those data are arranged). The term "relational database" is sometimes informally used to refer to a relational database management system, which is the software that is used to create and use a relational database.

The term relational database was originally defined and coined by Edgar Codd at IBM Almaden Research Center in 1970Contents

Strictly, a relational database is a collection of relations (frequently called tables). Other items are frequently considered part of the database, as they help to organize and structure the data, in addition to forcing the database to conform to a set of requirements.

Terminology

Relational database terminology.

Relational database theory uses a different set of mathematical-based terms, which are equivalent, or roughly equivalent, to SQL database terminology. The table below summarizes some of the most important relational database terms and their SQL database equivalents.

Relational term	**SQL equivalent**
relation, base relvar	table
derived relvar view,	query result, result set
tuple row	
attribute column	

Relations or Tables

A relation is defined as a set of tuples that have the same attributes A tuple usually represents an object and information about that object. Objects are typically physical objects or concepts. A relation is usually described as a table, which is organized into rows and columns. All the data referenced by an attribute are in the same domain and conform to the same constraints.

The relational model specifies that the tuples of a relation have no specific order and that the tuples, in turn, impose no order on the attributes. Applications access data by specifying queries, which use operations such as select to identify tuples, project to identify attributes, and join to combine relations. Relations can be modified using the *insert, delete, and update operators. New tuples can supply explicit* values or be derived from a query. Similarly, queries identify tuples for updating or deleting.

Base and Derived Relations

In a relational database, all data are stored and accessed via relations. Relations that store data are called "base relations", and in implementations are called "tables". Other relations do not store data, but are computed by applying relational operations to other relations. These relations are sometimes called "derived relations". In implementations these are called "views" or "queries". Derived relations are convenient in that though they may grab information from several relations, they act as a single relation. Also, derived relations can be used as an abstraction layer.

Keys

A unique key is a kind of constraint that ensures that an object, or critical information about the object, occurs in at most one tuple in a given relation. For example, a school might want each student to have a separate locker. To ensure this, the database designer creates a key on the locker attribute of the student relation. Keys can include more than one attribute, for example, a nation may impose a restriction that no province can have two cities with the same name. The key would include province and city name. This would still allow two different provinces to have a town called Springfield because their province is different. A key over more than one attribute is called a compound key.

Foreign Keys

A foreign key is a reference to a key in another relation, meaning that the referencing tuple has, as one of its attributes, the values of a key in the referenced tuple. Foreign keys need not have unique values in the referencing relation. Foreign keys effectively use the values of attributes in the referenced relation to restrict the domain of one or more attributes in the referencing relation.

A foreign key could be described formally as: "For all tuples in the referencing relation projected over the referencing attributes, there must exist a tuple in the referenced relation projected over those same attributes such that the values in each of the referencing attributes match the corresponding values in the referenced attributes."

4.0 CONCLUSION

Database applications are used to store and manipulate data. A database application can be used in many business functions including sales and inventory tracking, accounting, employee benefits, payroll, production and more. Database programs for personal computers come in various shape and sizes. A database remains fundamental for the implementation of any DATABASE MANAGEMENT APPLICATION SYSTEM.

5.0 SUMMARY

- A Database is a structured collection of data that is managed to meet the needs of a community of users. The structure is achieved by organizing the data according to a database model
- The earliest known use of the term database was in November 1963, when the System Development Corporation sponsored a symposium under the title Development and Management of a Computer-*centered Data Base.*
- Considering development in information technology and business applications have resulted in the evolution of several major types of databases.
- Database tables/indexes are typically stored in memory or on hard disk in one of many forms, ordered/unordered Flat files, ISAM, Heaps, Hash buckets or B+ Trees
- A database server is a computer program that provides database services to other computer programs or computers, as defined by the client-server model
- Database replication can be used on many database management systems, usually with a master/slave relationship between the original and the copies
- A relational database is a database that conforms to the relational model, and refers to a database's data and schema

6.0 TUTOR-MARKED ASSIGNMENT

1. Define the terms: Field, Records, Field Relation and Attribute
2. Briefly describe a flat file

7.0 REFERENCES/FURTHER READINGS

Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387. doi: 10.1145/362384.362685.

O’Brien, James A. (2003). (11th Edition) Introduction to Information Systems. McGraw-Hill.

UNIT 3 DATABASE CONCEPTS

CONTENTS

1.0 INTRODUCTION

There are basic and standard concepts associated with all databases, and these are what we will discuss in much detail in this unit. These include the concept of Creating, Reading, Updating and Deleting (CRUD) data, ACID (Atomicity, Consistency, Isolation, Durability), and Keys of different kinds.

2.0 OBJECTIVES

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

- know the meaning of the acronymn CRUD
- understand the applications of databases
- know the meaning of the acronymn ACID and how each members of the ACID differ from each other
- understand the structure of a database
- know the types of keys associated with databases.

3.0 MAIN CONTENT

3.1 Create, Read, Update and Delete

Create, read, update and delete (CRUD) are the four basic functions of persistent storage a major part of nearly all computer software. Sometimes CRUD is expanded with the words retrieve instead of read or destroys instead of delete. It is also sometimes used to describe user interface conventions that facilitate viewing, searching, and changing information; often using computer-based forms and reports.

Alternate terms for CRUD (one initialism and three acronyms):

•ABCD: add, browse, change, delete
•ACID: add, change, inquire, delete — though this can be confused with the transactional use of the acronym ACID.
•BREAD: browse, read, edit, add, delete
•VADE(R): view, add, delete, edit (and restore, for systems supporting transaction processing)

Database Applications

The acronym CRUD refers to all of the major functions that need to be implemented in a relational database application to consider it complete. Each letter in the acronym can be mapped to a standard SQL statement:

Operation	SQL
Create	INSERT
Read (Retrieve)	SELECT
Update	UPDATE
Delete (Destroy)	DELETE

Although a relational database is a common persistence layer in software applications, there are numerous others. CRUD can be implemented with an object database, an XML database, flat text files, custom file formats, tape, or card, for example.

Google Scholar lists the first reference to create-read-update-delete as by Kilov in 1990. The concept seems to be also described in more detail in Kilov's 1998 book.

User Interface

CRUD is also relevant at the user interface level of most applications. For example, in address book software, the basic storage unit is an individual contact entry. As a bare minimum, the software must allow the user to:

•Create or add new entries
•Read, retrieve, search, or view existing entries
•Update or edit existing entries
•Delete existing entries

Without at least these four operations, the software cannot be considered complete. Because these operations are so fundamental, they are often documented and described under one comprehensive heading, such as "contact management" or "contact maintenance" (or "document management" in general, depending on the basic storage unit for the particular application).

3.2 ACID

In computer science, ACID (Atomicity, Consistency, Isolation, *Durability) is a set of properties that guarantee that database transactions* are processed reliably. In the context of databases, a single logical operation on the data is called a transaction.

An example of a transaction is a transfer of funds from one account to another, even though it might consist of multiple individual operations (such as debiting one account and crediting another).

Atomicity

Atomicity refers to the ability of the DBMS to guarantee that either all of the tasks of a transaction are performed or none of them are. For example, the transfer of funds can be completed or it can fail for a multitude of reasons, but atomicity guarantees that one account won't be debited if the other is not credited. Atomicity states that database modifications must follow an "all or nothing" rule. Each transaction is said to be "atomic." If one part of the transaction fails, the entire transaction fails. It is critical that the database management system maintain the atomic nature of transactions in spite of any DBMS, operating system or hardware failure.

Consistency

Consistency property ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over (whether successful or not).

Consistency states that only valid data will be written to the database. If, for some reason, a transaction is executed that violates the database's consistency rules, the entire transaction will be rolled back and the database will be restored to a state consistent with those rules. On the other hand, if a transaction successfully executes, it will take the database from one state that is consistent with the rules to another state that is also consistent with the rules.

Durability

Durability refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone. This means it will survive system failure, and that the database system has checked the integrity constraints and won't need to abort the transaction. Many databases implement durability by writing all transactions into a log that can be played back to recreate the system state right before the failure. A transaction can only be deemed committed after it is safely in the log.

Implementation

Implementing the ACID properties correctly is not simple. Processing a transaction often requires a number of small changes to be made, including updating indices that are used by the system to speed up searches. This sequence of operations is subject to failure for a number of reasons; for instance, the system may have no room left on its disk drives, or it may have used up its allocated CPU time.

ACID suggests that the database be able to perform all of these operations at once. In fact this is difficult to arrange. There are two popular families of techniques: write ahead logging and shadow paging. In both cases, locks must be acquired on all information that is updated, and depending on the implementation, on all data that is being read. In write ahead logging, atomicity is guaranteed by ensuring that information about all changes is written to a log before it is written to the database. That allows the database to return to a consistent state in the event of a crash. In shadowing, updates are applied to a copy of the database, and the new copy is activated when the transaction commits. The copy refers to unchanged parts of the old version of the database, rather than being an entire duplicate.

Until recently almost all databases relied upon locking to provide ACID capabilities. This means that a lock must always be acquired before processing data in a database, even on read operations. Maintaining a large number of locks, however, results in substantial overhead as well as hurting concurrency. If user A is running a transaction that has read a row of data that user B wants to modify, for example, user B must wait until user A's transaction is finished.

An alternative to locking is multiversion concurrency control in which the database maintains separate copies of any data that is modified. This allows users to read data without acquiring any locks. Going back to the example of user A and user B, when user A's transaction gets to data that user B has modified, the database is able to retrieve the exact version of that data that existed when user A started their transaction. This ensures that user A gets a consistent view of the database even if other users are changing data that user A needs to read. A natural implementation of this idea results in a relaxation of the isolation property, namely snapshot isolation.

It is difficult to guarantee ACID properties in a network environment. Network connections might fail, or two users might want to use the same part of the database at the same time.

Two-phase commit is typically applied in distributed transactions to ensure that each participant in the transaction agrees on whether the transaction should be committed or not.

Care must be taken when running transactions in parallel. Two phase locking is typically applied to guarantee full isolation.

3.3 Keys

3.3.1 Foreign Key

In the context of relational databases, a foreign key is a referential constraint between two tables. The foreign key identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table. The columns in the referencing table must be the primary key or other candidate key in the referenced table. The values in one row of the referencing columns must occur in a single row in the referenced table. Thus, a row in the referencing table cannot contain values that don't exist in the referenced table (except potentially NULL). This way references can be made to link information together and it is an essential part of database normalization. Multiple rows in the referencing table may refer to the same row in the referenced table. Most of the time, it reflects the one (master table, or referenced table) to many (child table, or referencing table) relationship.

The referencing and referenced table may be the same table, i.e. the foreign key refers back to the same table. Such a foreign key is known in SQL:2003 as self-referencing or recursive foreign key.

A table may have multiple foreign keys, and each foreign key can have a different referenced table. Each foreign key is enforced independently by the database system. Therefore, cascading relationships between tables can be established using foreign keys.
Improper foreign key/primary key relationships or not enforcing those relationships are often the source of many database and data modeling problems.

Referential Actions

Because the DBMS enforces referential constraints, it must ensure data integrity if rows in a referenced table are to be deleted (or updated). If dependent rows in referencing tables still exist, those references have to be considered. SQL: 2003 specifies 5 different referential actions that shall take place in such occurrences:

•CASCADE
•RESTRICT
•NO ACTION
•SET NULL

•SET DEFAULT

CASCADE

Whenever rows in the master (referenced) table are deleted, the respective rows of the child (referencing) table with a matching foreign key column will get deleted as well. A foreign key with a cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted. This is called a cascade delete.

Example Tables: Customer(customer_id,cname,caddress)and Order(customer_id,products,payment)

Customer is the master table and Order is the child table, where 'customer_id' is the foreign key in Order and represents the customer who placed the order. When a row of Customer is deleted, any Order row matching the deleted Customer's customer_id will also be deleted. the values are deleted in the row like if we delete one row in the parent table then the same row in the child table will be automatically deleted.

RESTRICT

A row in the referenced table cannot be updated or deleted if dependent rows still exist. In that case, no data change is even attempted and should not be allowed.

NO ACTION

The UPDATE or DELETE SQL statement is executed on the referenced table. The DBMS verifies at the end of the statement execution if none of the referential relationships is violated. The major difference to RESTRICT is that triggers or the statement semantics itself may give a result in which no foreign key relationships is violated. Then, the statement can be executed successfully.

SET NULL

The foreign key values in the referencing row are set to NULL when the referenced row is updated or deleted. This is only possible if the respective columns in the referencing table are nullable. Due to the semantics of NULL, a referencing row with NULLs in the foreign key columns does not require a referenced row.

SET DEFAULT

Similarly to SET NULL, the foreign key values in the referencing row are set to the column default when the referenced row is updated or deleted.

3.3.2 Candidate Key

In the relational model, a candidate key of a relvar (relation variable) is a set of attributes of that relvar such that at all times it holds in the relation assigned to that variable that there are no two distinct turples with the same values for these attributes and there is not a proper subset of this set of attributes for which (1) holds.

Since a superkey is defined as a set of attributes for which (1) holds, we can also define a candidate key as a minimal superkey, i.e. a superkey of which no proper subset is also a superkey.

The importance of candidate keys is that they tell us how we can identify individual tuples in a relation. As such they are one of the most important types of database constraint that should be specified when designing a database schema. Since a relation is a set (no duplicate elements), it holds that every relation will have at least one candidate key (because the entire heading is always a superkey). Since in some RDBMSs tables may also represent multisets (which strictly means these DBMSs are not relational), it is an important design rule to specify explicitly at least one candidate key for each relation. For practical reasons RDBMSs usually require that for each relation one of its candidate keys is declared as the primary key, which means that it is considered as the preferred way to identify individual tuples. Foreign keys, for example, are usually required to reference such a primary key and not any of the other candidate keys.

Determining Candidate Keys

The previous example only illustrates the definition of candidate key and not how these are in practice determined. Since most relations have a large number or even infinitely many instances it would be impossible to determine all the sets of attributes with the uniqueness property for each instance. Instead it is easier to consider the sets of real-world entities that are represented by the relation and determine which attributes of the entities uniquely identify them. For example a relation *Employee(Name, Address, Dept) probably represents employees and* these are likely to be uniquely identified by a combination of Name and *Address which is therefore a superkey, and unless the same holds for* only Name or only Address, then this combination is also a candidate key.

In order to determine correctly the candidate keys it is important to determine all superkeys, which is especially difficult if the relation represents a set of relationships rather than a set of entities

3.3.3 Unique key

In relational database design, a unique key or primary key is a candidate key to uniquely identify each row in a table. A unique key or primary key comprises a single column or set of columns. No two distinct rows in a table can have the same value (or combination of values) in those columns. Depending on its design, a table may have arbitrarily many unique keys but at most one primary key.

A unique key must uniquely identify all possible rows that exist in a table and not only the currently existing rows. Examples of unique keys are Social Security numbers (associated with a specific person) or ISBNs (associated with a specific book). Telephone books and dictionaries cannot use names or words or Dewey Decimal system numbers as candidate keys because they do not uniquely identify telephone numbers or words.

A primary key is a special case of unique keys. The major difference is that for unique keys the implicit NOT NULL constraint is not automatically enforced, while for primary keys it is. Thus, the values in a unique key column may or may not be NULL. Another difference is that primary keys must be defined using another syntax.

The relational model, as expressed through relational calculus and relational algebra, does not distinguish between primary keys and other kinds of keys. Primary keys were added to the SQL standard mainly as a convenience to the application programmer.

Unique keys as well as primary keys can be referenced by form

3.3.4 Superkey

A superkey is defined in the relational model of database organization as a set of attributes of a relation variable (relvar) for which it holds that in all relations assigned to that variable there are no two distinct tuples (rows) that have the same values for the attributes in this set. Equivalently a superkey can also be defined as a set of attributes of a relvar upon which all attributes of the relvar are functionally dependent. Note that if attribute set K is a superkey of relvar R, then at all times it is the case that the projection of R over K has the same cardinality as R itself.

Informally, a superkey is a set of columns within a table whose values can be used to uniquely identify a row. A candidate key is a minimal set of columns necessary to identify a row, this is also called a minimal superkey. For example, given an employee table, consisting of the columns employeeID, name, job, and departmentID, we could use the employeeID in combination with any or all other columns of this table to uniquely identify a row in the table. Examples of superkeys in this table would be {employeeID, Name}, {employeeID, Name, job}, and {employeeID, Name, job, departmentID}.

In a real database we don't need values for all of those columns to identify a row. We only need, per our example, the set {employeeID}. This is a minimal superkey – that is, a minimal set of columns that can be used to identify a single row. So, employeeID is a candidate key.

Example

English Monarchs

Monarch Name	Monarch Number	Royal House
Edward II		Plantagenet
Edward III		Plantagenet
Richard II		Plantagenet
Henry IV		Lancaster

In this example, the possible superkeys are:

•{Monarch Name, Monarch Number}
•{Monarch Name, Monarch Number, Royal House}

3.3.4 Surrogate key

A surrogate key in a database is a unique identifier for either an entity in the modeled world or an object in the database. The surrogate key is *not derived from application data.*

Definition

There appear to be two definitions of a surrogate in the literature. We shall call these surrogate (1) and surrogate (2):

Surrogate (1)

This definition is based on that given by Hall, Owlett and Todd (1976). Here a surrogate represents an entity in the outside world. The surrogate is internally generated by the system but is nevertheless visible by the user or application.

Surrogate (2)

This definition is based on that given by Wieringa and de Jung (1991). Here a surrogate represents an object in the database itself. The surrogate is internally generated by the system and is invisible to the user or application.

We shall adopt the surrogate (1) definition throughout this article largely because it is more data model rather than storage model oriented. See Date (1998).

An important distinction exists between a surrogate and a primary key, depending on whether the database is a current database or a temporal database. A current database stores only currently valid data, therefore there is a one-to-one correspondence between a surrogate in the modelled world and the primary key of some object in the database; in this case the surrogate may be used as a primary key, resulting in the term surrogate key. However, in a temporal database there is a many-to-one relationship between primary keys and the surrogate. Since there may be several objects in the database corresponding to a single surrogate, we cannot use the surrogate as a primary key; another attribute is required, in addition to the surrogate, to uniquely identify each object.

Although Hall et alia (1976) say nothing about this, other authors have argued that a surrogate should have the following constraints:

- the value is unique system-wide, hence never reused;
- the value is system generated;
- the value is not manipulable by the user or application;
- the value contains no semantic meaning;
- the value is not visible to the user or application;
- the value is not composed of several values from different domains.

Surrogates in Practice

In a current database, the surrogate key can be the primary key, generated by the DATABASE MANAGEMENT APPLICATION SYSTEM and not derived from any application data in the database. The only significance of the surrogate key is to act as the primary key. It is also possible that the surrogate key exists in addition to the database-generated uuid, e.g. a HR number for each employee besides the UUID of each employee.

A surrogate key is frequently a sequential number (e.g. a Sybase or SQL Server "identity column", a PostgreSQL serial, an Oracle SEQUENCE or a column defined with AUTO_INCREMENT in MySQL) but doesn't have to be. Having the key independent of all other columns insulates the database relationships from changes in data values or database design (making the database more agile) and guarantees uniqueness.

In a temporal database, it is necessary to distinguish between the surrogate key and the primary key. Typically, every row would have both a primary key and a surrogate key. The primary key identifies the unique row in the database, the surrogate key identifies the unique entity in the modelled world; these two keys are not the same. For example, table Staff may contain two rows for "John Smith", one row when he was employed between 1990 and 1999, another row when he was employed between 2001 and 2006. The surrogate key is identical (non-unique) in both rows however the primary key will be unique.

Some database designers use surrogate keys religiously regardless of the suitability of other candidate keys, while others will use a key already present in the data, if there is one.
A surrogate may also be called a

- surrogate key,
- entity identifier,
- system-generated key,
- database sequence number,
- synthetic key,
- technical key, or
- arbitrary unique identifier.

Some of these terms describe the way of generating new surrogate values rather than the nature of the surrogate concept.

4.0 CONCLUSION

The fundamental concepts that guide the operation of a database, that is, CRUD and ACID remains the same irrespective of the types and models of databases that emerge by the day. However, one cannot rule out the possibilities of other concepts emerging with time in the near future.

5.0 SUMMARY

- Create, read, update and delete (CRUD) are the four basic functions of persistent storage a major part of nearly all computer software.
- In computer science, ACID (Atomicity, Consistency, Isolation, *Durability) is a set of properties that guarantee that database* transactions are processed reliably. In the context of databases, a single logical operation on the data is called a transaction.

- In the context of relational databases a foreign key is a referential constraint between two tables
- In the relational model, a candidate key of a relvar (relation variable) is a set of attributes of that relvar such that at all times it holds in the relation assigned to that variable that there are no two distinct tuples with the same values for these attributes
- In relational database design, a unique key or primary key is a candidate key to uniquely identify each row in a table
- Superkey: A superkey is defined in the relational model of database organization as a set of attributes of a relation variable (relvar) for which it holds that in all relations assigned to that variable there are no two distinct tuples (rows) that have the same values for the attributes in this set
- A surrogate key in a database is a unique identifier for either an *entity in the modeled world or an object in the database.*

6.0 TUTOR-MARKED ASSIGNMENT

1. What are the meaning of the acronyms CRUD and ACID
2. What are the constraints associated with surrogate keys

7.0 REFERENCES/FURTHER READINGS

Nijssen, G.M. (1976). Modelling in Data Base Management Systems. North-Holland Pub. Co. ISBN 0-7204-0459-2.

Engles, R.W.: (1972). A Tutorial on Data-Base Organization, Annual Review in Automatic Programming, Vol.7, Part 1, Pergamon Press, Oxford, pp. 1–64.

Langefors, B: (1968). Elementary Files and Elementary File Records, Proceedings of File 68, an IFIP/IAG International Seminar on File Organisation, Amsterdam, November, pp. 89–96.

The Identification of Objects and Roles: Object Identifiers Revisited by Wieringa and de Jung (1991).

Relational Database Writings 1994–1997 by C.J. Date (1998), Chapters 11 and 12.

Carter, Breck. "Intelligent Versus Surrogate Keys". Retrieved on 2006-12-03.

Richardson, Lee. "Create Data Disaster: Avoid Unique Indexes – (Mistake 3 of 10)".

Berkus, Josh. "Database Soup: Primary Keyvil, Part I".

Gray, Jim (September 1981). "The Transaction Concept: Virtues and Limitations". Proceedings of the 7th International Conference on *Very Large Data Bases: pages 144–154, 19333 Vallco Parkway,* Cupertino CA 95014: Tandem Computers.

Jim Gray & Andreas Reuter, Distributed Transaction Processing: Concepts and Techniques, Morgan Kaufman 1993. ISBN 1558601902.

Date, Christopher (2003). "5: Integrity", An Introduction to Database *Systems. Addison-Wesley, pp. 268-276. ISBN 978-0321189561*.

UNIT 4 DATABASE MODELS 1

CONTENTS

1.0 INTRODUCTION

Several models have evolved in the course of development of databases and DATABASE MANAGEMENT APPLICATION SYSTEM. This has resulted in several forms of models deployed by users depending on their needs and understanding. In this unit we set the pace to X-ray these models and conclude in subsequent unit.

2.0 OBJECTIVES

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

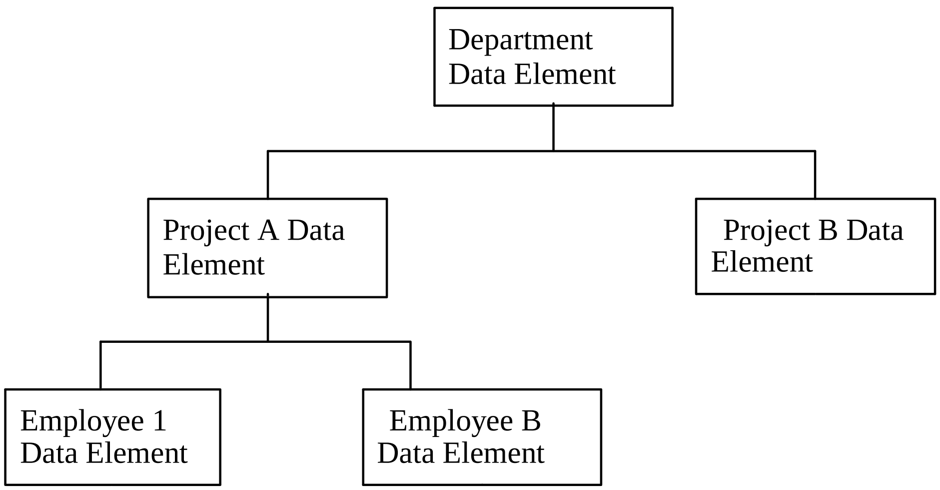
- know and define the different types of database models
- differentiate the database models from each other
- sketch the framework of hierarchical and network models
- understand the concepts and model behind the models
- know the advantages and disadvantages of the different models.

3.0 MAIN CONTENT

3.1 Hierarchical Model

In a hierarchical model, data is organized into an inverted tree-like 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. This structure arranges the various data elements in a hierarchy and helps to establish logical relationships among data elements of multiple files. Each unit in the model is a record which is also known as a node. In such a model, each record on one level can be related to multiple records on the next lower level. A record that has subsidiary records is called a parent and the subsidiary records are called children. Data elements in this model are well suited for one-to-many relationships with other data elements in the database.

Figure 1: A Hierarchical Structure



This model is advantageous when the data elements are inherently hierarchical. The disadvantage is that in order to prepare the database it becomes necessary to identify the requisite groups of files that are to be logically integrated. Hence, a hierarchical data model may not always be flexible enough to accommodate the dynamic needs of an organization.

Example

An example of a hierarchical data model would be if an organization had records of employees in a table (entity type) called "Employees". In the table there would be attributes/columns such as First Name, Last Name, Job Name and Wage. The company also has data about the employee's children in a separate table called "Children" with attributes such as First Name, Last Name, and date of birth. The Employee table represents a parent segment and the Children table represents a Child segment. These two segments form a hierarchy where an employee may have many children, but each child may only have one parent.

Consider the following structure:

EmpNo	Designation	ReportsTo
10	Director	
20	Senior Manager	10
30	Typist	20
40	Programmer	20

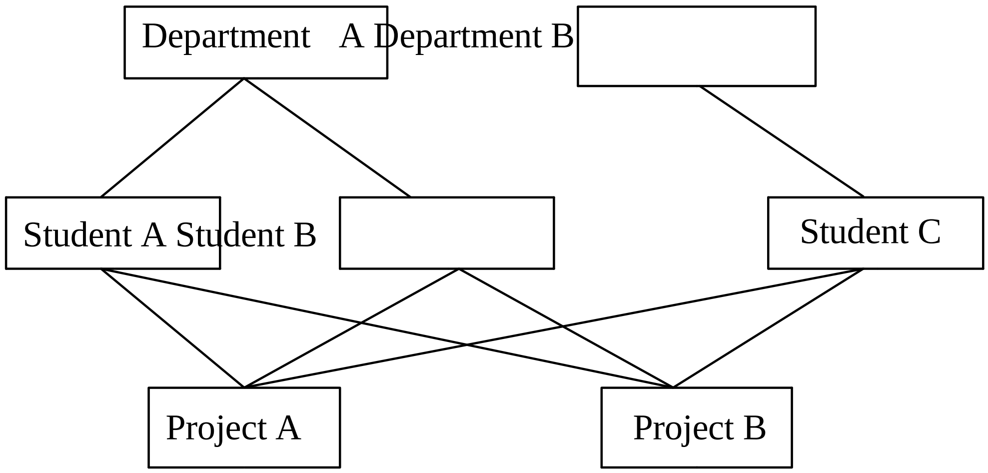
In this, the "child" is the same type as the "parent". The hierarchy stating EmpNo 10 is boss of 20, and 30 and 40 each report to 20 is represented by the "ReportsTo" column. In Relational database terms, the ReportsTo column is a foreign key referencing the EmpNo column. If the "child" data type were different, it would be in a different table, but there would still be a foreign key referencing the EmpNo column of the employees table.

This simple model is commonly known as the adjacency list model, and was introduced by Dr. Edgar F. Codd after initial criticisms surfaced that the relational model could not model hierarchical data.

3.2 Network Model

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). These relationships (somewhat confusingly) are called sets. For example a student might be a member of one set whose owner is the course they are studying, and a member of another set whose owner is the college they belong to. At the same time the student might be the owner of a set of email addresses, and owner of another set containing phone numbers. The main difference between the network model and hierarchical model is that in a network model, a child can have a number of parents whereas in a hierarchical model, a child can have only one parent. The hierarchical model is therefore a subset of the network model.

Figure 3: Network Structure



Programmatic access to network databases is traditionally by means of a navigational data manipulation language, in which programmers navigate from a current record to other related records using verbs such as find owner, find next, and find prior. The most common example of such an interface is the COBOL-based Data Manipulation Language defined by CODASYL.

Network databases are traditionally implemented by using chains of pointers between related records. These pointers can be node numbers or disk addresses.

The network model became popular because it provided considerable flexibility in modelling complex data relationships, and also offered high performance by virtue of the fact that the access verbs used by programmers mapped directly to pointer-following in the implementation.

The network model provides greater advantage than the hierarchical model in that it promotes greater flexibility and data accessibility, since records at a lower level can be accessed without accessing the records above them. This model is more efficient than hierarchical model, easier to understand and can be applied to many real world problems that require routine transactions. The disadvantages are that: It is a complex process to design and develop a network database; It has to be refined frequently; It requires that the relationships among all the records be defined before development starts, and changes often demand major programming efforts; Operation and maintenance of the network model is expensive and time consuming.

Examples of database engines that have network model capabilities are RDM Embedded and RDM Server.

However, the model had several disadvantages. Networkl programming proved error-prone as data models became more complex, and small changes to the data structure could require changes to many programs. Also, because of the use of physical pointers, operations such as database loading and restructuring could be very time-consuming.

Concept and History: The network model is a database model conceived as a flexible way of representing objects and their relationships. Its original inventor was Charles Bachman, and it was developed into a standard specification published in 1969 by the CODASYL Consortium. Where the hierarchical model structures data as a tree of records, with each record having one parent record and many children, the network model allows each record to have multiple parent and child records, forming a lattice structure.

The chief argument in favour of the network model, in comparison to the hierarchic model, was that it allowed a more natural modeling of relationships between entities. Although the model was widely implemented and used, it failed to become dominant for two main reasons. Firstly, IBM chose to stick to the hierarchical model with semi-network extensions in their established products such as IMS and DL/I. Secondly, it was eventually displaced by the relational model, which offered a higher-level, more declarative interface. Until the early 1980s the performance benefits of the low-level navigational interfaces offered by hierarchical and network databases were persuasive for many large-scale applications, but as hardware became faster, the extra productivity and flexibility of the relational model led to the gradual obsolescence of the network model in corporate enterprise usage.

3.3 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 systems 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.

3.4 Object Database

In an object database (also object oriented database), information is represented in the form of objects as used in object-oriented programming. When database capabilities are combined with object programming language capabilities, the result is an object database management system (ODBMS). An ODBMS makes database objects appear as programming language objects in one or more object programming languages. An ODBMS extends the programming language with transparently persistent data, concurrency control, data recovery, associative queries, and other capabilities.

Some object-oriented databases are designed to work well with object-oriented programming languages such as Python, Java, C#, Visual Basic .NET, C++, Objective-C and Smalltalk. Others have their own programming languages. An ODBMSs use exactly the same model as object-oriented programming languages.

Object databases are generally recommended when there is a business need for high performance processing on complex data.

Adoption of Object Databases

Object databases based on persistent programming acquired a niche in application areas such as engineering and spatial databases, telecommunications, and scientific areas such as high energy physics and molecular biology. They have made little impact on mainstream commercial data processing, though there is some usage in specialized areas of financial service]. It is also worth noting that object databases held the record for the World's largest database (being first to hold over 1000 Terabytes at Stanford Linear Accelerator Center "Lessons Learned From Managing A Petabyte") and the highest ingest rate ever recorded for a commercial database at over one Terabyte per hour.

Another group of object databases focuses on embedded use in devices, packaged software, and realtime systems.

Advantages and Disadvantages

Benchmarks between ODBMSs and RDBMSs have shown that an ODBMS can be clearly superior for certain kinds of tasks. The main reason for this is that many operations are performed using navigational rather than declarative interfaces, and navigational access to data is usually implemented very efficiently by following pointers.

Critics of navigational database-based technologies like ODBMS suggest that pointer-based techniques are optimized for very specific "search routes" or viewpoints. However, for general-purpose queries on the same information, pointer-based techniques will tend to be slower and more difficult to formulate than relational. Thus, navigation appears to simplify specific known uses at the expense of general, unforeseen, and varied future uses. However, with suitable language support, direct object references may be maintained in addition to normalised, indexed aggregations, allowing both kinds of access; furthermore, a persistent language may index aggregations on whatever is returned by some arbitrary object access method, rather than only on attribute value, which can simplify some queries.

Other things that work against an ODBMS seem to be the lack of interoperability with a great number of tools/features that are taken for granted in the SQL world including but not limited to industry standard connectivity, reporting tools, OLAP tools, and backup and recovery standards. Additionally, object databases lack a formal mathematical foundation, unlike the relational model, and this in turn leads to weaknesses in their query support. However, this objection is offset by the fact that some ODBMSs fully support SQL in addition to navigational access, e.g. Objectivity/SQL++, Matisse, and InterSystems CACHÉ. Effective use may require compromises to keep both paradigms in sync.

In fact there is an intrinsic tension between the notion of encapsulation, which hides data and makes it available only through a published set of interface methods, and the assumption underlying much database technology, which is that data should be accessible to queries based on data content rather than predefined access paths. Database-centric thinking tends to view the world through a declarative and attribute-driven viewpoint, while OOP tends to view the world through a behavioral viewpoint, maintaining entity-identity independently of changing attributes. This is one of the many impedance mismatch issues surrounding OOP and databases.

Although some commentators have written off object database technology as a failure, the essential arguments in its favor remain valid, and attempts to integrate database functionality more closely into object programming languages continue in both the research and the industrial communities.

3.5 Associative Model of Data

The associative model of data is an alternative data model for database systems. Other data models, such as the relational model and the object data model, are record-based. These models involve encompassing attributes about a thing, such as a car, in a record structure. Such attributes might be registration, colour, make, model, etc. In the associative model, everything which has "discrete independent existence" is modeled as an entity, and relationships between them are modeled as associations. The granularity at which data is represented is similar to schemes presented by Chen (Entity-relationship model); Bracchi, Paolini and Pelagatti (Binary Relations); and Senko (The Entity Set Model).

3.6 Column-Oriented DBMS

A column-oriented DBMS is a DATABASE MANAGEMENT APPLICATION SYSTEM (DBMS) which stores its content by column rather than by row. This has advantages for databases such as data warehouses and library catalogues, where aggregates are computed over large numbers of similar data items.

Benefits

Comparisons between row-oriented and column-oriented systems are typically concerned with the efficiency of hard-disk access for a given workload, as seek time is incredibly long compared to the other delays in computers. Further, because seek time is improving at a slow rate relative to cpu power (see Moore's Law), this focus will likely continue on systems reliant on hard-disks for storage. Following is a set of over-simplified observations which attempt to paint a picture of the trade-offs between column and row oriented organizations.

1. Column-oriented systems are more efficient when an aggregate needs to be computed over many rows but only for a notably smaller subset of all columns of data, because reading that smaller subset of data can be faster than reading all data.

2. Column-oriented systems are more efficient when new values of a column are supplied for all rows at once, because that column data can be written efficiently and replace old column data without touching any other columns for the rows.

3. Row-oriented systems are more efficient when many columns of a single row are required at the same time, and when row-size is relatively small, as the entire row can be retrieved with a single disk seek.

4. Row-oriented systems are more efficient when writing a new row if all of the column data is supplied at the same time, as the entire row can be written with a single disk seek.

In practice, row oriented architectures are well-suited for OLTP-like workloads which are more heavily loaded with interactive transactions. Column stores are well-suited for OLAP-like workloads (e.g., data warehouses) which typically involve a smaller number of highly complex queries over all data (possibly terabytes).

Storage Efficiency vs. Random Access

Column data is of uniform type; therefore, there are some opportunities for storage size optimizations available in column oriented data that are not available in row oriented data. For example, many popular modern compression schemes, such as LZW, make use of the similarity of adjacent data to compress. While the same techniques may be used on row-oriented data, a typical implementation will achieve less effective results. Further, this behavior becomes more dramatic when a large percentage of adjacent column data is either the same or not-present, such as in a sparse column (similar to a sparse matrix). The opposing tradeoff is Random Access. Retrieving all data from a single row is more efficient when that data is located in a single location, such as in a row-oriented architecture. Further, the greater adjacent compression achieved, the more difficult random-access may become, as data might need to be uncompressed to be read.

Implementations

For many years, only the Sybase IQ product was commonly available in the column-oriented DBMS class. However, that has changed rapidly in the last few years with many open source and commercial implementations.

3.7 Navigational Database

Navigational databases are characterized by the fact that objects in the database are found primarily by following references from other objects. Traditionally navigational interfaces are procedural, though one could characterize some modern systems like XPath as being simultaneously navigational and declarative.

Navigational access is traditionally associated with the network model and hierarchical model of database interfaces and have evolved into Set-oriented systems. Navigational techniques use "pointers" and "paths" to navigate among data records (also known as "nodes"). This is in contrast to the relational model (implemented in relational databases), which strives to use "declarative" or logic programming techniques in which you ask the system for what you want instead of how to navigate to it.

For example, to give directions to a house, the navigational approach would resemble something like, "Get on highway 25 for 8 miles, turn onto Horse Road, left at the red barn, then stop at the 3rd house down the road". Whereas, the declarative approach would resemble, "Visit the green house(s) within the following coordinates...."

Hierarchical models are also considered navigational because one "goes" up (to parent), down (to leaves), and there are "paths", such as the familiar file/folder paths in hierarchical file systems. In general, navigational systems will use combinations of paths and prepositions such as "next", "previous", "first", "last", "up", "down", etc.

Some also suggest that navigational database engines are easier to build and take up less memory (RAM) than relational equivalents. However, the existence of relational or relational-based products of the late 1980s that possessed small engines (by today's standards) because they did not use SQL suggest this is not necessarily the case. Whatever the reason, navigational techniques are still the preferred way to handle smaller-scale structures.

A current example of navigational structuring can be found in the Document Object Model (DOM) often used in web browsers and closely associated with JavaScript. The DOM "engine" is essentially a light-weight navigational database. The World Wide Web itself and Wikipedia could even be considered forms of navigational databases. (On a large scale, the Web is a network model and on smaller or local scales, such as domain and URL partitioning, it uses hierarchies.)

3.8 Distributed Database

A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.

Collections of data (e.g. in a database) can be distributed across multiple physical locations. A distributed database is distributed into separate partitions/fragments. Each partition/fragment of a distributed database may be replicated (i.e. redundant fail-overs, RAID like).

Besides distributed database replication and fragmentation, there are many other distributed database design technologies. For example, local autonomy, synchronous and asynchronous distributed database technologies. These technologies' implementation can and does depend on the needs of the business and the sensitivity/confidentiality of the data to be stored in the database, and hence the price the business is willing to spend on ensuring data security, consistency and integrity.

Important considerations

Care with a distributed database must be taken to ensure the following:

- The distribution is transparent — users must be able to interact with the system as if it were one logical system. This applies to the system's performance, and methods of access amongst other things.
- Transactions are transparent — each transaction must maintain database integrity across multiple databases. Transactions must also

be divided into subtransactions, each subtransaction affecting one database system.

Advantages of Distributed Databases

- Reflects organizational structure — database fragments are located in the departments they relate to.
- Local autonomy — a department can control the data about them (as they are the ones familiar with it.)
- Improved availability — a fault in one database system will only affect one fragment, instead of the entire database.
- Improved performance — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the databases to be balanced among servers. (A high load on one module of the database won't affect other modules of the database in a distributed database.)
- Economics — it costs less to create a network of smaller computers with the power of a single large computer.
- Modularity — systems can be modified, added and removed from the distributed database without affecting other modules (systems).

Disadvantages of Distributed Databases

- Complexity — extra work must be done by the DBAs to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.
- Economics — increased complexity and a more extensive infrastructure means extra labour costs.
- Security — remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (e.g., by encrypting the network links between remote sites).
- Difficult to maintain integrity — in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.
- Inexperience — distributed databases are difficult to work with, and as a young field there is not much readily available experience on proper practice.
- Lack of standards – there are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS.

- Database design more complex – besides of the normal difficulties, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication.

3.9 Real Time Database

A real-time database is a processing system designed to handle workloads whose state is constantly changing (Buchmann). This differs from traditional databases containing persistent data, mostly unaffected by time. For example, a stock market changes very rapidly and is dynamic. The graphs of the different markets appear to be very unstable and yet a database has to keep track of current values for all of the markets of the New York Stock Exchange (Kanitkar). Real-time processing means that a transaction is processed fast enough for the result to come back and be acted on right away (Capron). Real-time databases are useful for accounting, banking, law, medical records, multi-media, process control, reservation systems, and scientific data analysis (Snodgrass). As computers increase in power and can store more data, they are integrating themselves into our society and are employed in many applications.

Overview

Real-time databases are traditional databases that use an extension to give the additional power to yield reliable responses. They use timing constraints that represent a certain range of values for which the data are valid. This range is called temporal validity. A conventional database cannot work under these circumstances because the inconsistencies between the real world objects and the data that represents them are too severe for simple modifications. An effective system needs to be able to handle time-sensitive queries, return only temporally valid data, and support priority scheduling. To enter the data in the records, often a sensor or an input device monitors the state of the physical system and updates the database with new information to reflect the physical system more accurately (Abbot). When designing a real-time database system, one should consider how to represent valid time, how facts are associated with real-time system. Also, consider how to represent attribute values in the database so that process transactions and data consistency have no violations (Abbot).

When designing a system, it is important to consider what the system should do when deadlines are not met. For example, an air-traffic control system constantly monitors hundreds of aircraft and makes decisions about incoming flight paths and determines the order in which aircraft should land based on data such as fuel, altitude, and speed. If any of this information is late, the result could be devastating (Sivasankaran). To address issues of obsolete data, the timestamp can support transactions by providing clear time references (Sivasankaran).

SQL DBMS

IBM started working on a prototype system loosely based on Codd's concepts as System R in the early 1970s — unfortunately, System R was conceived as a way of proving Codd's ideas unimplementable, and thus the project was delivered to a group of programmers who were not under Codd's supervision, never understood his ideas fully and ended up violating several fundamentals of the relational model. The first "quickie" version was ready in 1974/5, and work then started on multi-table systems in which the data could be broken down so that all of the data for a record (much of which is often optional) did not have to be stored in a single large "chunk". Subsequent multi-user versions were tested by customers in 1978 and 1979, by which time a standardized query language, SQL, had been added. Codd's ideas were establishing themselves as both workable and superior to Codasyl, pushing IBM to develop a true production version of System R, known as SQL/DS, and, later, Database 2 (DB2).

Many of the people involved with INGRES became convinced of the future commercial success of such systems, and formed their own companies to commercialize the work but with an SQL interface. Sybase, Informix, NonStop SQL and eventually Ingres itself were all being sold as offshoots to the original INGRES product in the 1980s. Even Microsoft SQL Server is actually a re-built version of Sybase, and thus, INGRES. Only Larry Ellison's Oracle started from a different chain, based on IBM's papers on System R, and beat IBM to market when the first version was released in 1978.

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

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

4.0 CONCLUSION

The evolution of database models is continuous until a time an ideal model will emerge that will meet all the requirements of end users. This sound impossible because there can never be a system that is completely fault-free. Thus we will yet see more of models of database. The flat and hierarchical models had set the tune for emerging models.

5.0 SUMMARY

- In a hierarchical model, data is organized into an inverted tree-like 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.
- 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).
- 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 an object database (also object oriented database), information is represented in the form of objects as used in object-oriented programming.
- The associative model of data is an alternative data model for database systems. Other data models, such as the relational model and the object data model, are record-based.
- A column-oriented DBMS is a database management system (DBMS) which stores its content by column rather than by row. This has advantages for databases such as data warehouses and library catalogues, where aggregates are computed over large numbers of similar data items
- Navigational databases are characterized by the fact that objects in the database are found primarily by following references from other objects.
- A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU
- A real-time database is a processing system designed to handle workloads whose state is constantly changing (Buchmann). This differs from traditional databases containing persistent data, mostly unaffected by time

6.0 TUTOR-MARKED ASSIGNMENT

1. Mention 5 models of databases

2. Briefly discuss the advantages and disadvantages of distributed databases

7.0 REFERENCES/FURTHER READINGS

Charles W. Bachman, The Programmer as Navigator. ACM Turing Award Lecture, Communications of the ACM, Volume 16, Issue 11, 1973, pp. 653-658, ISSN 0001-0782, doi: 10.1145/355611.362534.

Stonebraker, Michael with Moore, Dorothy. Object-Relational DBMSs: *The Next Great Wave. Morgan Kaufmann Publishers, 1996.* ISBN 1-55860-397-2.

There was, at the Time, Some Dispute Whether the Term was coined by Michael Stonebraker of Illustra or Won Kim of UniSQL.

Kim, Won. Introduction to Object-Oriented Databases. The MIT Press, 1990. ISBN 0-262-11124-1.

Bancilhon, Francois; Delobel, Claude; and Kanellakis, Paris. Building *an Object-Oriented Database System: The Story of O2. Morgan* Kaufmann Publishers, 1992. ISBN 1-55860-169-4.

C-Store: A column-oriented DBMS, Stonebraker et al, Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

Blazewicz, Jacek; Królikowski, Zbyszko; Morzy, Tadeusz (2003). *Handbook on Data Management in Information Systems.* Springer, pp. 18. ISBN 3540438939.

M. T. Ozsu and P. Valduriez, Principles of Distributed Databases (2nd edition), Prentice-Hall, ISBN 0-13-659707-6 Federal Standard 1037C.

Elmasri and Navathe, Fundamentals of Database Systems (3rd edition), Addison-Wesley Longman, ISBN 0-201-54263-3.

Abbot, Robert K., and Hector Garcia-Molina. Scheduling Real-Time Transactions: a Performance Evaluation. Stanford University and Digital Equipment Corp. ACM, 1992. 13 Dec. 2006 .

Buchmann, A. "Real Time Database Systems." Encyclopedia of Database Technologies and Applications. Ed. Laura C. Rivero, Jorge H. Doorn, and Viviana E. Ferraggine. Idea Group, 2005.

Stankovic, John A., Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms. Springer, 1998.

UNIT 5 DATABASE MODELS: RELATIONAL MODEL

CONTENTS

3.0 Main Content
- 3.1 The Model
- 3.2 Interpretation
- 3.3 Application to Databases
- 3.4 Alternatives to the Relational Model
- 3.5 History
- 3.6 SQL and the Relational Model
- 3.7 Implementation
- 3.8 Controversies
- 3.9 Design
- **3.10 Set-Theoretic Formulation**
- **3.11 Key Constraints and Functional Dependencies**

4.0 Conclusion
5.0 Summary
6.0 Tutor-Marked Assignment
7.0 References/Further Readings

1.0 INTRODUCTION

The relational model for database management is a database model based on first-order predicate logic, first formulated and proposed in 1969 by Edgar Codd

Its core idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values. The content of the database at any given time is a finite model (logic) of the database, i.e. a set of **relations, one per predicate variable, such that all predicates are** satisfied. A request for information from the database (a database query) is also a predicate.

The purpose of the relational model is to provide a declarative method for specifying data and queries: we directly state what information the database contains and what information we want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for getting queries answered.

IBM implemented Codd's ideas with the DB2 database management system; it introduced the SQL data definition and query language. Other relational database management systems followed, most of them using SQL as well. A table in an SQL database schema corresponds to a predicate variable; the contents of a table to a relation; key constraints, other constraints, and SQL queries correspond to predicates. However, it must be noted that SQL databases, including DB2, deviate from the relational model in many details; Codd fiercely argued against deviations that compromise the original principles.]

2.0 OBJECTIVES

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

- define relational model of database
- understand and explain the concept behind relational models
- answer the question of how to interpret a relational database model
- know the various applications of relational database
- compare relational model with the structured query language (SQL)
- know the constraints and controversies associated with relational database model.

Figure 1: Relational Structure

Department Table

Deptno	**Dname**	**Dloc**	**Dmgr**
Dept A			
Dept B			
Dept C			

Employee Table

Empno	**Ename**	**Etitle**	**Esalary**	**Deptno**
Emp 1				**Dept A**
Emp 2				**Dept B**
Emp 3				**Dept C**
Emp 4				**Dept D**
Emp 5				**Dept E**
Emp 6				**Dept F**

3.0 MAIN CONTENT

3.1 The Model

The fundamental assumption of the relational model is that all data is represented as mathematical n-ary relations, an n-ary relation being a subset of the Cartesian product of n domains. In the mathematical model, reasoning about such data is done in two-valued predicate logic, meaning there are two possible evaluations for each proposition: either *true or false (and in particular no third value such as unknown, or not applicable, either of which are often associated with the concept of* NULL). Some think two-valued logic is an important part of the relational model, where others think a system that uses a form of three-valued logic can still be considered relational]

Data are operated upon by means of a relational calculus or relational algebra, these being equivalent in expressive power.

The relational model of data permits the database designer to create a consistent, logical representation of information. Consistency is achieved by including declared constraints in the database design, which is usually referred to as the logical schema. The theory includes a process of database normalization whereby a design with certain desirable properties can be selected from a set of logically equivalent alternatives. The access plans and other implementation and operation details are handled by the DBMS engine, and are not reflected in the logical model. This contrasts with common practice for SQL DBMSs in which performance tuning often requires changes to the logical model. The basic relational building block is the domain or data type, usually abbreviated nowadays to type. A tuple is an unordered set of attribute ***values. An attribute is an ordered pair of attribute name and type name.*** An attribute value is a specific valid value for the type of the attribute. This can be either a scalar value or a more complex type.

A relation consists of a heading and a body. A heading is a set of attributes. A body (of an n-ary relation) is a set of n-tuples. The heading of the relation is also the heading of each of its tuples.

A relation is defined as a set of n-tuples. In both mathematics and the relational database model, a set is an unordered collection of items, although some DBMSs impose an order to their data. In mathematics, a tuple has an order, and allows for duplication. E.F. Codd originally defined tuples using this mathematical definition. Later, it was one of E.F. Codd’s great insights that using attribute names instead of an ordering would be so much more convenient (in general) in a computer language based on relations. This insight is still being used today. Though the concept has changed, the name "tuple" has not. An immediate and important consequence of this distinguishing feature is that in the relational model the Cartesian product becomes commutative. A table is an accepted visual representation of a relation; a tuple is similar to the concept of row, but note that in the database language SQL the columns and the rows of a table are ordered.

A relvar is a named variable of some specific relation type, to which at all times some relation of that type is assigned, though the relation may contain zero tuples.

The basic principle of the relational model is the Information Principle: all information is represented by data values in relations. In accordance with this Principle, a relational database is a set of relvars and the result of every query is presented as a relation.

The consistency of a relational database is enforced, not by rules built into the applications that use it, but rather by constraints, declared as part of the logical schema and enforced by the DBMS for all applications. In general, constraints are expressed using relational comparison operators, of which just one, "is subset of" ( ), is theoretically sufficient. In practice, several useful shorthands are expected to be available, of which the most important are candidate key (really, superkey) and foreign key constraints.

3.2 Interpretation

To fully appreciate the relational model of data it is essential to understand the intended interpretation of a relation.

The body of a relation is sometimes called its extension. This is because it is to be interpreted as a representation of the extension of some predicate, this being the set of true propositions that can be formed by replacing each free variable in that predicate by a name (a term that designates something).

There is a one-to-one correspondence between the free variables of the predicate and the attribute names of the relation heading. Each tuple of the relation body provides attribute values to instantiate the predicate by substituting each of its free variables. The result is a proposition that is deemed, on account of the appearance of the tuple in the relation body, to be true. Contrariwise, every tuple whose heading conforms to that of the relation but which does not appear in the body is deemed to be false. This assumption is known as the closed world assumption

For a formal exposition of these ideas, see the section Set Theory **Formulation, below.**

3.3 Application to Databases

A type as used in a typical relational database might be the set of integers, the set of character strings, the set of dates, or the two boolean values true and false, and so on. The corresponding type names for these types might be the strings "int", "char", "date", "boolean", etc. It is important to understand, though, that relational theory does not dictate what types are to be supported; indeed, nowadays provisions are expected to be available for user-defined types in addition to the built-in ones provided by the system.

Attribute is the term used in the theory for what is commonly referred to as a column. Similarly, table is commonly used in place of the theoretical term relation (though in SQL the term is by no means synonymous with relation). A table data structure is specified as a list of column definitions, each of which specifies a unique column name and the type of the values that are permitted for that column. An attribute **value is the entry in a specific column and row, such as "John Doe" or** "35".

A tuple is basically the same thing as a row, except in an SQL DBMS, where the column values in a row are ordered. (Tuples are not ordered; instead, each attribute value is identified solely by the attribute name and never by its ordinal position within the tuple.) An attribute name might be "name" or "age".

A relation is a table structure definition (a set of column definitions) along with the data appearing in that structure. The structure definition is the heading and the data appearing in it is the body, a set of rows. A database relvar (relation variable) is commonly known as a base table. The heading of its assigned value at any time is as specified in the table declaration and its body is that most recently assigned to it by invoking some update operator (typically, INSERT, UPDATE, or DELETE). The heading and body of the table resulting from evaluation of some query are determined by the definitions of the operators used in the expression of that query. (Note that in SQL the heading is not always a set of column definitions as described above, because it is possible for a column to have no name and also for two or more columns to have the same name. Also, the body is not always a set of rows because in SQL it is possible for the same row to appear more than once in the same body.)

3.4 Alternatives to the Relational Model

Other models are the hierarchical model and network model. Some systems using these older architectures are still in use today in data centers with high data volume needs or where existing systems are so complex and abstract it would be cost prohibitive to migrate to systems employing the relational model; also of note are newer object-oriented databases, even though many of them are DBMS-construction kits, rather than proper DBMSs.

A recent development is the Object-Relation type-Object model, which is based on the assumption that any fact can be expressed in the form of one or more binary relationships. The model is used in Object Role Modeling (ORM), RDF/Notation 3 (N3) and in Gellish English.

The relational model was the first formal database model. After it was defined, informal models were made to describe hierarchical databases (the hierarchical model) and network databases (the network model). Hierarchical and network databases existed before relational databases, but were only described as models after the relational model was defined, in order to establish a basis for comparison.

3.5 History

The relational model was invented by E.F. (Ted) Codd as a general model of data, and subsequently maintained and developed by Chris Date and Hugh Darwen among others. In The Third Manifesto (first published in 1995) Date and Darwen show how the relational model can accommodate certain desired object-oriented features.

3.6 SQL and the Relational Model

SQL, initially pushed as the standard language for relational databases, deviates from the relational model in several places. The current ISO SQL standard doesn't mention the relational model or use relational terms or concepts. However, it is possible to create a database conforming to the relational model using SQL if one does not use certain SQL features.

The following deviations from the relational model have been noted in SQL. Note that few database servers implement the entire SQL standard and in particular do not allow some of these deviations. Whereas NULL is nearly ubiquitous, for example, allowing duplicate column names within a table or anonymous columns is uncommon.

Duplicate Rows

The same row can appear more than once in an SQL table. The same tuple cannot appear more than once in a relation.

Anonymous Columns

A column in an SQL table can be unnamed and thus unable to be referenced in expressions. The relational model requires every attribute to be named and referenceable.

Duplicate Column Names

Two or more columns of the same SQL table can have the same name and therefore cannot be referenced, on account of the obvious ambiguity. The relational model requires every attribute to be referenceable.

Column Order Significance

The order of columns in an SQL table is defined and significant, one consequence being that SQL's implementations of Cartesian product and union are both noncommutative. The relational model requires that there should be of no significance to any ordering of the attributes of a relation.

Views without CHECK OPTION

Updates to a view defined without CHECK OPTION can be accepted but the resulting update to the database does not necessarily have the expressed effect on its target. For example, an invocation of INSERT can be accepted but the inserted rows might not all appear in the view, or an invocation of UPDATE can result in rows disappearing from the view. The relational model requires updates to a view to have the same effect as if the view were a base relvar.

Columnless Tables Unrecognized

SQL requires every table to have at least one column, but there are two relations of degree zero (of cardinality one and zero) and they are needed to represent extensions of predicates that contain no free variables.

NULL

This special mark can appear instead of a value wherever a value can appear in SQL, in particular in place of a column value in some row. The deviation from the relational model arises from the fact that the implementation of this ad hoc concept in SQL involves the use of three-valued logic, under which the comparison of NULL with itself does not yield true but instead yields the third truth value, unknown; similarly the comparison NULL with something other than itself does not yield false but instead yields unknown. It is because of this behaviour in comparisons that NULL is described as a mark rather than a value. The relational model depends on the law of excluded middle under which anything that is not true is false and anything that is not false is true; it also requires every tuple in a relation body to have a value for every attribute of that relation. This particular deviation is disputed by some if only because E.F. Codd himself eventually advocated the use of special marks and a 4-valued logic, but this was based on his observation that there are two distinct reasons why one might want to use a special mark in place of a value, which led opponents of the use of such logics to discover more distinct reasons and at least as many as 19 have been noted, which would require a 21-valued logic. SQL itself uses NULL for several purposes other than to represent "value unknown". For example, the sum of the empty set is NULL, meaning zero, the average of the empty set is NULL, meaning undefined, and NULL appearing in the result of a LEFT JOIN can mean "no value because there is no matching row in the right-hand operand".

Concepts

SQL uses concepts "table", "column", "row" instead of "relvar", "attribute", "tuple". These are not merely differences in terminology. For example, a "table" may contain duplicate rows, whereas the same tuple cannot appear more than once in a relation.

3.7 Implementation

There have been several attempts to produce a true implementation of the relational database model as originally defined by Codd and explained by Date, Darwen and others, but none have been popular successes so far. Rel is one of the more recent attempts to do this.

3.8 Controversies

Codd himself, some years after publication of his 1970 model, proposed a three-valued logic (True, False, Missing or NULL) version of it in order to deal with missing information, and in his The Relational Model *for Database Management Version 2 (1990) he went a step further with* a four-valued logic (True, False, Missing but Applicable, Missing but Inapplicable) version. But these have never been implemented, presumably because of attending complexity. SQL's NULL construct was intended to be part of a three-valued logic system, but fell short of that due to logical errors in the standard and in its implementations.

3.9 Design

Database normalization is usually performed when designing a relational database, to improve the logical consistency of the database design. This trades off transactional performance for space efficiency.

There are two commonly used systems of diagramming to aid in the visual representation of the relational model: the entity-relationship diagram (ERD), and the related IDEF diagram used in the IDEF1X method created by the U.S. Air Force based on ERDs.

The tree structure of data may enforce hierarchical model organization, with parent-child relationship table.

3.10 Set-Theoretic Formulation

Basic notions in the relational model are relation names and attribute *names. We will represent these as strings such as "Person" and "name"* and we will usually use the variables and a,b,c to range over them. Another basic notion is the set of atomic values that contains values such as numbers and strings.

Our first definition concerns the notion of tuple, which formalizes the notion of row or record in a table:

Tuple

A tuple is a partial function from attribute names to atomic values.
Header

A header is a finite set of attribute names.
Projection

The projection of a tuple t on a finite set of attributes A is.

The next definition defines relation which formalizes the contents of a table as it is defined in the relational model.

Relation

A relation is a tuple (H,B) with H, the header, and B, the body, a set of tuples that all have the domain H.

Such a relation closely corresponds to what is usually called the extension of a predicate in first-order logic except that here we identify the places in the predicate with attribute names. Usually in the relational model a database schema is said to consist of a set of relation names, the headers that are associated with these names and the constraints that should hold for every instance of the database schema.

3.11 Key Constraints and Functional Dependencies

One of the simplest and most important types of relation constraints is the key constraint. It tells us that in every instance of a certain relational schema the tuples can be identified by their values for certain attributes.

4.0 CONCLUSION

The evolution of the relational model of database and database management systems is significant in the history and development of database and DATABASE MANAGEMENT APPLICATION SYSTEMs. This concept pioneered by Edgar Codd brought an entirely and much efficient way of storing and retrieving data, especially for a large database. This concept emphasized the use of tables and then linking the tables through commands. Most of today's DATABASE MANAGEMENT APPLICATION SYSTEMs implements the relational model

5.0 SUMMARY

- The relational model for database management is a database model based on first-order predicate logic, first formulated and proposed in 1969 by Edgar Codd
- The fundamental assumption of the relational model is that all data is represented as mathematical n-ary relations, an n-ary relation being a subset of the Cartesian product of n domains.
- To fully appreciate the relational model of data it is essential to understand the intended interpretation of a relation.
- A type as used in a typical relational database might be the set of integers, the set of character strings, the set of dates, or the two boolean values true and false, and so on
- Other models are the hierarchical model and network model. Some systems using these older architectures are still in use today in data centers
- The relational model was invented by E.F. (Ted) Codd as a general model of data, and subsequently maintained and developed by Chris Date and Hugh Darwen among others.
- SQL, initially pushed as the standard language for relational databases, deviates from the relational model in several places.
- There have been several attempts to produce a true implementation of the relational database model as originally defined by Codd and explained by Date, Darwen and others, but none have been popular successes so far

- Database normalization is usually performed when designing a relational database, to improve the logical consistency of the database design
- Basic notions in the relational model are relation names and *attribute names.*
- One of the simplest and most important types of relation constraints is the key constraint.

6.0 TUTOR-MARKED ASSIGNMENT

1. Briefly discuss Interpretation in Relational Model.
2. Mention 5 ways in which relational model differs from an SQL

7.0 REFERENCES/FURTHER READINGS

"Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks", E.F. Codd, IBM Research Report, 1969.

"A Relational Model of Data for Large Shared Data Banks", in Communications of the ACM, 1970.

White, Colin. In the Beginning: An RDBMS History. Teradata Magazine Online. September 2004 edition. URL: http://www.teradata.com/t/page/127057.

Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387. doi: 10.1145/362384.362685.

Date, C. J., Darwen, H. (2000). Foundation for Future Database *Systems: The Third Manifesto, 2nd edition, Addison-Wesley* Professional. ISBN 0-201-70928-7.

Date, C. J. (2003). Introduction to Database Systems. 8th edition, Addison-Wesley. ISBN 0-321-19784-4.

UNIT 6 BASIC COMPONENTS OF DBMS

CONTENTS

1.0 INTRODUCTION

To be discussed in these units are the basic components of any database. These components ensure proper control of data, access of data, query for data as well as methods of accessing DATABASE MANAGEMENT APPLICATION SYSTEMs.

2.0 OBJECTIVES

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

- know the rules guiding transaction ACID
- know what is concurrency control in databases
- mention the different methods of concurrency control
- define and interpret the acronymn JDBC
- answer the question of the types and drivers of JDBC
- define query optimizer, and its applications and cost estimation

3.0 MAIN CONTENT

3.1 Concurrency Controls

In databases, concurrency control ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible.

Concurrency Control in Databases

Concurrency control in DATABASE MANAGEMENT APPLICATION SYSTEMs (DBMS) ensures that database transactions are performed concurrently without the concurrency violating the data integrity of a database. Executed transactions should follow the ACID rules, as described below. The DBMS must guarantee that only serializable (unless Serializability is intentionally relaxed), recoverable schedules are generated. It also guarantees that no effect of committed transactions is lost, and no effect of aborted (rolled back) transactions remains in the related database.

Transaction ACID Rules

•Atomicity - Either the effects of all or none of its operations remain when a transaction is completed - in other words, to the outside world the transaction appears to be indivisible, atomic.
•Consistency - Every transaction must leave the database in a consistent state.
•Isolation - Transactions cannot interfere with each other. Providing isolation is the main goal of concurrency control.
•Durability - Successful transactions must persist through crashes.

Concurrency Control Mechanism

The main categories of concurrency control mechanisms are:

•Optimistic - Delay the synchronization for a transaction until it is end without blocking (read, write) operations, and then abort transactions that violate desired synchronization rules.
•Pessimistic - Block operations of transaction that would cause violation of synchronization rules.

There are several methods for concurrency control. Among them:

•Two-phase locking
•Strict two-phase locking
•Conservative two-phase locking
•Index locking
•Multiple granularity locking

A Lock is a database system object associated with a database object (typically a data item) that prevents undesired (typically synchronization rule violating) operations of other transactions by blocking them. Database system operations check for lock existence, and halt when noticing a lock type that is intended to block them.

There are also non-lock concurrency control methods, among them:

•Conflict (serializability, precedence) graph checking
•Timestamp ordering

•commitment ordering
•Also Optimistic concurrency control methods typically do not use locks.

Almost all currently implemented lock-based and non-lock-based concurrency control mechanisms guarantee schedules that are conflict serializable (unless relaxed forms of serializability are needed). However, there are many research texts encouraging view serializable schedules for possible gains in performance, especially when not too many conflicts exist (and not too many aborts of completely executed transactions occur), due to reducing the considerable overhead of blocking mechanisms.

Concurrency Control in Operating Systems

Operating systems, especially real-time operating systems, need to maintain the illusion that many tasks are all running at the same time. Such multitasking is fairly simple when all tasks are independent from each other. However, when several tasks try to use the same resource, or when tasks try to share information, it can lead to confusion and inconsistency. The task of concurrent computing is to solve that problem. Some solutions involve "locks" similar to the locks used in databases, but they risk causing problems of their own such as deadlock. Other solutions are lock-free and wait-free algorithms.

3.2 Java Database Connectivity

Java Database Connectivity (JDBC) is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.

Overview

JDBC has been part of the Java Standard Edition since the release of JDK 1.1. The JDBC classes are contained in the Java package java.sql. Starting with version 3.0, JDBC has been developed under the Java Community Process. JSR 54 specifies JDBC 3.0 (included in J2SE 1.4), JSR 114 specifies the JDBC Rowset additions, and JSR 221 is the specification of JDBC 4.0 (included in Java SE 6).

JDBC allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager. The Driver Manager is used as a connection factory for creating JDBC connections.

JDBC connections support creating and executing statements. These may be update statements such as SQL's CREATE, INSERT, UPDATE and DELETE, or they may be query statements such as SELECT. Additionally, stored procedures may be invoked through a JDBC connection. JDBC represents statements using one of the following classes:

- •Statement – the statement is sent to the database server each and every time.
- •PreparedStatement – the statement is cached and then the execution path is pre determined on the database server allowing it to be executed multiple times in an efficient manner.
- •CallableStatement – used for executing stored procedures on the database.

Update statements such as INSERT, UPDATE and DELETE return an update count that indicates how many rows were affected in the database. These statements do not return any other information.

Query statements return a JDBC row result set. The row result set is used to walk over the result set. Individual columns in a row are retrieved either by name or by column number. There may be any number of rows in the result set. The row result set has metadata that describes the names of the columns and their types.

There is an extension to the basic JDBC API in the javax.sql package that allows for scrollable result sets and cursor support among other things.

JDBC Drivers

JDBC Drivers are client-side adaptors (they are installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand.

Types: There are commercial and free drivers available for most relational database servers. These drivers fall into one of the following types:

- •Type 1,the JDBC-ODBC bridge
- •Type 2, the Native-API driver
- •Type 3, the network-protocol driver
- •Type 4 the native-protocol drivers

Internal JDBC driver, driver embedded with JRE in Java-enabled SQL databases. Used for Java stored procedures. This does not belong to the above classification, although it would likely be either a type 2 or type 4 driver (depending on whether the database itself is implemented in Java or not). An example of this is the KPRB driver supplied with Oracle RDBMS. "jdbc:default:connection" is a relatively standard way of referring making such a connection (at least Oracle and Apache Derby support it). The distinction here is that the JDBC client is actually running as part of the database being accessed, so access can be made directly rather than through network protocols.

Sources

- •SQLSummit.com publishes list of drivers, including JDBC drivers and vendors
- •Sun Microsystems provides a list of some JDBC drivers and vendors
- •Simba Technologies ships an SDK for building custom JDBC Drivers for any custom/proprietary relational data source
- •DataDirect Technologies provides a comprehensive suite of fast Type 4 JDBC drivers for all major database
- •IDS Software provides a Type 3 JDBC driver for concurrent access to all major databases. Supported features include resultset caching, SSL encryption, custom data source, dbShield.
- •i-net software provides fast Type 4 JDBC drivers for all major databases
- •OpenLink Software ships JDBC Drivers for a variety of databases, including Bridges to other data access mechanisms (e.g., ODBC, JDBC) which can provide more functionality than the targeted mechanism
- •JDBaccess is a Java persistence library for MySQL and Oracle which defines major database access operations in an easy usable API above JDBC
- •JNetDirect provides a suite of fully Sun J2EE certified high performance JDBC drivers.
- •HSQLis a RDBMS with a JDBC driver and is available under a BSD license.

3.3 Query Optimizer

The query optimizer is the component of a database management system that attempts to determine the most efficient way to execute a query. The optimizer considers the possible query plans for a given input query, and attempts to determine which of those plans will be the most efficient. Cost-based query optimizers assign an estimated "cost" to each possible query plan, and choose the plan with the smallest cost.

Costs are used to estimate the runtime cost of evaluating the query, in terms of the number of I/O operations required, the CPU requirements, and other factors determined from the data dictionary. The set of query plans examined is formed by examining the possible access paths (e.g. index scan, sequential scan) and join algorithms (e.g. sort-merge join, hash join, nested loops). The search space can become quite large depending on the complexity of the SQL query.

The query optimizer cannot be accessed directly by users. Instead, once queries are submitted to database server, and parsed by the parser, they are then passed to the query optimizer where optimization occurs.

Implementation

Most query optimizers represent query plans as a tree of "plan nodes". A plan node encapsulates a single operation that is required to execute the query. The nodes are arranged as a tree, in which intermediate results flow from the bottom of the tree to the top. Each node has zero or more child nodes -- those are nodes whose output is fed as input to the parent node. For example, a join node will have two child nodes, which represent the two join operands, whereas a sort node would have a single child node (the input to be sorted). The leaves of the tree are nodes which produce results by scanning the disk, for example by performing an index scan or a sequential scan.

Cost Estimation

One of the hardest problems in query optimization is to accurately estimate the costs of alternative query plans. Optimizers cost query plans using a mathematical model of query execution costs that relies heavily on estimates of the cardinality, or number of tuples, flowing through each edge in a query plan. Cardinality estimation in turn depends on estimates of the selection factor of predicates in the query. Traditionally, database systems estimate selectivities through fairly detailed statistics on the distribution of values in each column, such as histograms This technique works well for estimation of selectivities of individual predicates. However many queries have conjunctions of predicates such as select count (*) from R where R.make='Honda' and R.model='Accord'. Query predicates are often highly correlated (for example, model='Accord' implies make='Honda'), and it is very hard to estimate the selectivity of the conjunct in general. Poor cardinality estimates and uncaught correlation are one of the main reasons why query optimizers pick poor query plans. This is one reason why a DBA should regularly update the database statistics, especially after major data loads/unloads.

3.4 Open Database Connectivity

In computing, Open Database Connectivity (ODBC) provides a standard software API method for using DATABASE MANAGEMENT APPLICATION SYSTEMs (DBMS). The designers of ODBC aimed to make it independent of programming languages, database systems, and operating systems.

Overview

The PRATAP specification offers a procedural API for using SQL queries to access data. An implementation of ODBC will contain one or more applications, a core ODBC "Driver Manager" library, and one or more "database drivers". The Driver Manager, independent of the applications and DBMS, acts as an "interpreter" between the applications and the database drivers, whereas the database drivers contain the DBMS-specific details. Thus a programmer can write applications that use standard types and features without concern for the specifics of each DBMS that the applications may encounter. Likewise, database driver implementors need only know how to attach to the core library. This makes ODBC modular.

To write ODBC code that exploits DBMS-specific features requires more advanced programming: an application must use introspection, calling ODBC metadata functions that return information about supported features, available types, syntax, limits, isolation levels, driver capabilities and more. Even when programmers use adaptive techniques, however, ODBC may not provide some advanced DBMS features. The ODBC 3.x API operates well with traditional SQL applications such as OLTP, but it has not evolved to support richer types introduced by SQL: 1999 and SQL:2003

ODBC provides the standard of ubiquitous data access because hundreds of ODBC drivers exist for a large variety of data sources. ODBC operates with a variety of operating systems and drivers exist for non-relational data such as spreadsheets, text and XML files. Because ODBC dates back to 1992, it offers connectivity to a wider variety of data sources than other data-access APIs. More drivers exist for ODBC than drivers or providers exist for newer APIs such as OLE DB, JDBC, and ADO.NET.

Despite the benefits of ubiquitous connectivity and platform-independence, systems designers may perceive ODBC as having certain drawbacks. Administering a large number of client machines can involve a diversity of drivers and DLLs. This complexity can increase system-administration overhead. Large organizations with thousands of PCs have often turned to ODBC server technology (also known as "Multi-Tier ODBC Drivers") to simplify the administration problems.

Differences between drivers and driver maturity can also raise important issues. Newer ODBC drivers do not always have the stability of drivers already deployed for years. Years of testing and deployment mean a driver may contain fewer bugs.

Developers needing features or types not accessible with ODBC can use other SQL APIs. When not aiming for platform-independence, developers can use proprietary APIs, whether DBMS-specific (such as TransactSQL) or language-specific (for example: JDBC for Java applications).

Bridging configurations

JDBC-ODBC Bridges

A JDBC-ODBC bridge consists of a JDBC driver which employs an ODBC driver to connect to a target database. This driver translates JDBC method calls into ODBC function calls. Programmers usually use such a bridge when a particular database lacks a JDBC driver. Sun Microsystems included one such bridge in the JVM, but viewed it as a stop-gap measure while few JDBC drivers existed. Sun never intended its bridge for production environments, and generally recommends against its use. Independent data-access vendors now deliver JDBC-ODBC bridges which support current standards for both mechanisms, and which far outperform the JVM built-in.

ODBC-JDBC Bridges

An ODBC-JDBC bridge consists of an ODBC driver which uses the services of a JDBC driver to connect to a database. This driver translates ODBC function calls into JDBC method calls. Programmers usually use such a bridge when they lack an ODBC driver for a particular database but have access to a JDBC driver.

Implementations

ODBC implementations run on many operating systems, including Microsoft Windows, Unix, Linux, OS/2, OS/400, IBM i5/OS, and Mac OS X. Hundreds of ODBC drivers exist, including drivers for Oracle, DB2, Microsoft SQL Server, Sybase, Pervasive SQL, IBM Lotus Domino, MySQL, PostgreSQL, and desktop database products such as FileMaker, and Microsoft Access.

3.5 Data Dictionary

A data dictionary, as defined in the IBM Dictionary of Computing is a "centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format. The term may have one of several closely related meanings pertaining to databases and DATABASE MANAGEMENT APPLICATION SYSTEMs (DBMS):

- a document describing a database or collection of databases
- an integral component of a DBMS that is required to determine its structure
- a piece of middleware that extends or supplants the native data dictionary of a DBMS

Data Dictionary Documentation

Database users and application developers can benefit from an authoritative data dictionary document that catalogs the organization, contents, and conventions of one or more databases This typically includes the names and descriptions of various tables and fields in each database, plus additional details, like the type and length of each data element. There is no universal standard as to the level of detail in such a document, but it is primarily a distillation of metadata about database structure, not the data itself. A data dictionary document also may include further information describing how data elements are encoded. One of the advantages of well-designed data dictionary documentation is that it helps to establish consistency throughout a complex database, or across a large collection of federated databases

Data Dictionary Middleware

In the construction of database applications, it can be useful to introduce an additional layer of data dictionary software, i.e. middleware, which communicates with the underlying DBMS data dictionary. Such a "high-level" data dictionary may offer additional features and a degree of flexibility that goes beyond the limitations of the native "low-level" data dictionary, whose primary purpose is to support the basic functions of the DBMS, not the requirements of a typical application. For example, a high-level data dictionary can provide alternative entity-relationship models tailored to suit different applications that share a common database. Extensions to the data dictionary also can assist in query optimization against distributed databases

Software frameworks aimed at rapid application development sometimes include high-level data dictionary facilities, which can substantially reduce the amount of programming required to build menus, forms, reports, and other components of a database application, including the database itself. For example, PHPLens includes a PHP class library to automate the creation of tables, indexes, and foreign key constraints portably for multiple databases. Another PHP-based data dictionary, part of the RADICORE toolkit, automatically generates program objects, scripts, and SQL code for menus and forms with data validation and complex JOINs For the ASP.NET environment, Base One's data dictionary provides cross-DBMS facilities for automated database creation, data validation, performance enhancement (caching and index utilization), application security, and extended data types.

4.0 CONCLUSION

The basic components of any database management system serve to ensure the availability of data as well as the efficiency in accessing the data. They include mainly, a data dictionary, query optimizers, and Java database connectivity.

5.0 SUMMARY

- In databases, concurrency control ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible.
- Java Database Connectivity (JDBC) is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.
- The query optimizer is the component of a database management system that attempts to determine the most efficient way to execute a query. The optimizer considers the possible query plans for a given input query, and attempts to determine which of those plans will be the most efficient.
- In computing, Open Database Connectivity (ODBC) provides a standard software API method for using database management systems (DBMS). The designers of ODBC aimed to make it independent of programming languages, database systems, and operating systems.
- A data dictionary, as defined in the IBM Dictionary of Computing is a "centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format
- In the construction of database applications, it can be useful to introduce an additional layer of data dictionary software, i.e. middleware, which communicates with the underlying DBMS data dictionary

6.0 TUTOR-MARKED ASSIGNMENT

1. Define the Transaction ACID rules.
2. List and define types of JDBC Driver.

7.0 REFERENCES/FURTHER READINGS

ACM, IBM Dictionary of Computing, 10th edition, 1993

TechTarget, SearchSOA, What is a Data Dictionary?

AHIMA Practice Brief, Guidelines for Developing a Data Dictionary, *Journal of AHIMA 77, no.2 (February 2006): 64A-D.*

U.S. Patent 4774661, Database management system with active data dictionary, 11/19/1985, AT&T

U.S. Patent 4769772, Automated Query Optimization Method using both Global and Parallel Local Optimizations for Materialization access Planning for Distributed Databases, 02/28/1985, Honeywell Bull.

PHPLens, ADOdb Data Dictionary Library for PHP

RADICORE, what is a Data Dictionary?

Base One International Corp., Base One Data Dictionary

Chaudhuri, Surajit (1998). "An Overview of Query Optimization in Relational Systems”. Proceedings of the ACM Symposium on *Principles of Database Systems: pages 34–43. doi:* 10.1145/275487.275492.

Ioannidis, Yannis (March 1996). "Query optimization". ACM *Computing Surveys 28 (1): 121–123. doi:* 10.1145/234313.234367.

Selinger, Patricia, et al. (1979). "Access Path Selection in a Relational Database Management System". Proceedings of the 1979 ACM *SIGMOD International Conference on Management of Data:* 23-34. doi:10.1145/582095.582099.

Parkes, Clara H. (April 1996). "Power to the People", DBMS Magazine, Miller Freeman, Inc.

MODULE 2

UNIT 1 DEVELOPMENT AND DESIGN-OF DATABASE

CONTENTS

1.0 INTRODUCTION

Database design is the process of deciding how to organize data into recordstypes and how the record types and how the record types and howthe record types will relate to each other. The DBMS mirror's the organization's data structure and process transactions efficiently.

Developing small, personal databases is relatively easy using microcomputer DBMS packages or wizards. However, developing a large database of complex of complex data types can be a complex task.
In many companies, developing and managing large corporate databases are the primary responsibility of the database administrator and database design analysts. They work with end users and systems analyst to model business processes and the data required. Then they determine:

1. What data definitions should be included in the databases
2. What structures or relationships should exist among the data elements?

2.0 OBJECTIVES

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

- understand the concept of data planning and database design
- know the steps in the development of databases
- identify the functions of each step of the design process
- define database normalization
- know the problems addressed by normalizations

• define normal forms from 1st to 6th forms
• define and understand the term denormalization

3.0 MAIN CONTENT

3.1 Database Development

3.1.1 Data Planning and Database Design

As figure 1 illustrates, database development may start with a top-down **data planning process. Database administrators and designers work** with corporate and end user management to develop an enterprise **model that defines the basic business process of the enterprise. Then** they define the information needs of end-users in a business process such as the purchasing/ receiving process that all business has.

Next, end users must identify the key data elements that are needed to perform the specific business activities. This frequently involves developing entity relationships among the diagrams (ERDs) that model the relationships among the many entities involved in the business processes. End users and database designers could use ERD available to identify what suppliers and product data are required to activate their purchasing/receiving and other business processes using enterprise resource planning (ERP) or supply chain management (SCM) software.

Such users' views are a major part of a data modeling process where the relationships between data elements are identified. Each data model defines the logical relationships among the data elements needed to support a basic business process. For example, can a supplier provide more than the type of product to use? Can a customer have more than one type of product to use? Can a customer have more than one type of account with us? Can an employee have several pay rates or be assigned to several projects or workgroup?

Answering such questions will identify data relationships that have to be represented in a data model that supports a business process. These data models then serves as logical frameworks (called schemas and sub schemas) on which to base the physical design of databases and the development of application programs to support business processes of the organization. A schema is an overall logical view of the relationship among the data elements in a database, while the sub schema is a logical view of the data relationships needed to support specific end user application programs that will access that database.

Remember that data models represent logical views of data and relationships of the database. Physical database design takes a physical *view of the data (also called internal view) that describes how data are to* be physically stored and accessed on the storage devices of a computer system. For example, figure 2 illustrates these different views and the software interface of a bank database processing system. In this example, checking, saving and installment lending are the business process where data models are part of a banking services data model that serves as a logical data framework for all bank services.

3.2 Design of Database

3.2.1 Database Normalization

Sometimes referred to as canonical synthesis, is a technique for designing relational database tables to minimize duplication of information and, in so doing, to safeguard the database against certain types of logical or structural problems, namely data anomalies. For example, when multiple instances of a given piece of information occur in a table, the possibility exists that these instances will not be kept consistent when the data within the table is updated, leading to a loss of data integrity. A table that is sufficiently normalized is less vulnerable to problems of this kind, because its structure reflects the basic assumptions for when multiple instances of the same information should be represented by a single instance only.

Higher degrees of normalization typically involve more tables and create the need for a larger number of joins, which can reduce performance. Accordingly, more highly normalized tables are typically used in database applications involving many isolated transactions (e.g. an Automated teller machine), while less normalized tables tend to be used in database applications that need to map complex relationships between data entities and data attributes (e.g. a reporting application, or a full-text search application).

Database theory describes a table's degree of normalization in terms of normal forms of successively higher degrees of strictness. A table in Third Normal Form (3NF), for example, is consequently in Second Normal Form (2NF) as well; but the reverse is not necessarily the case.

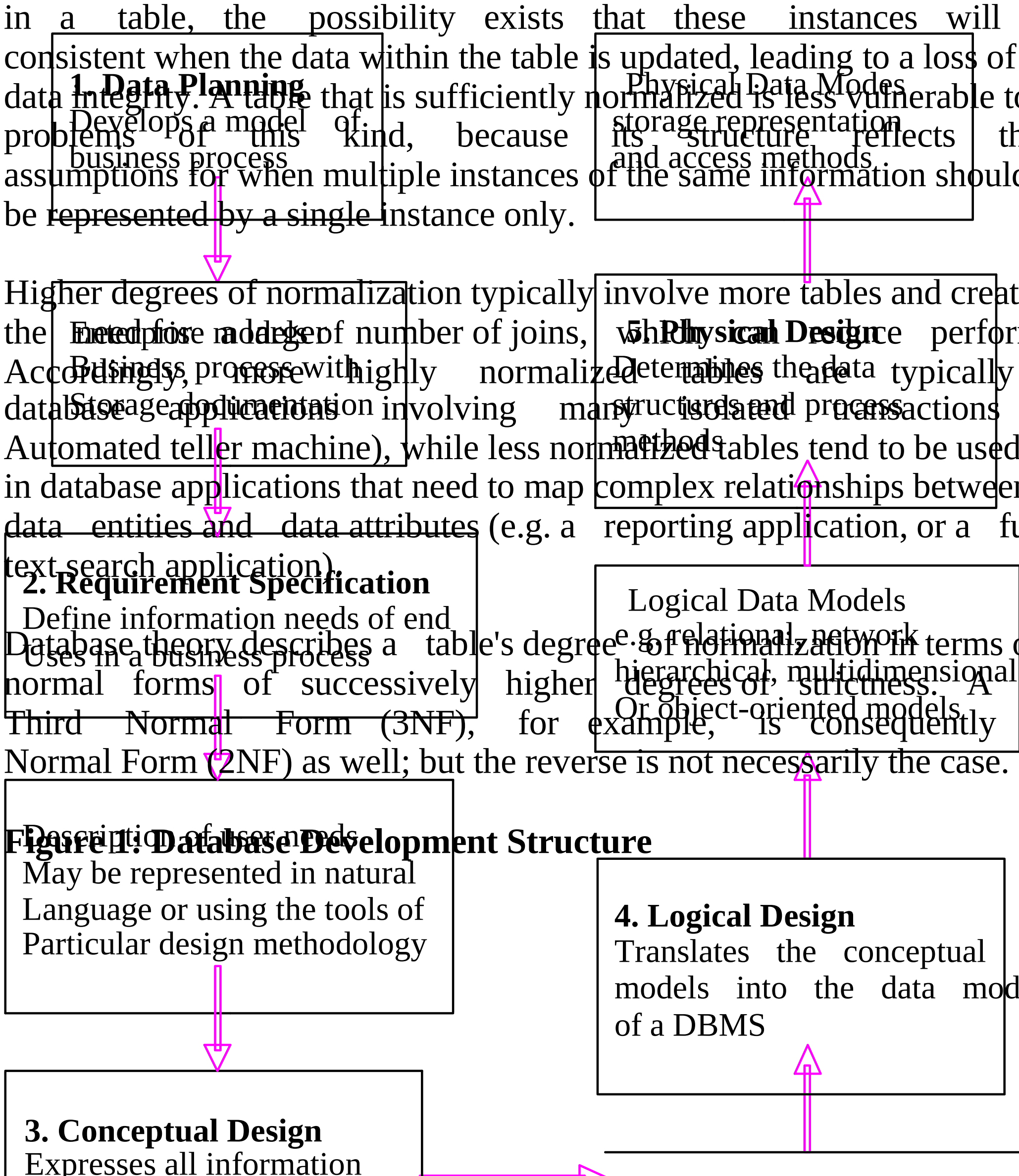
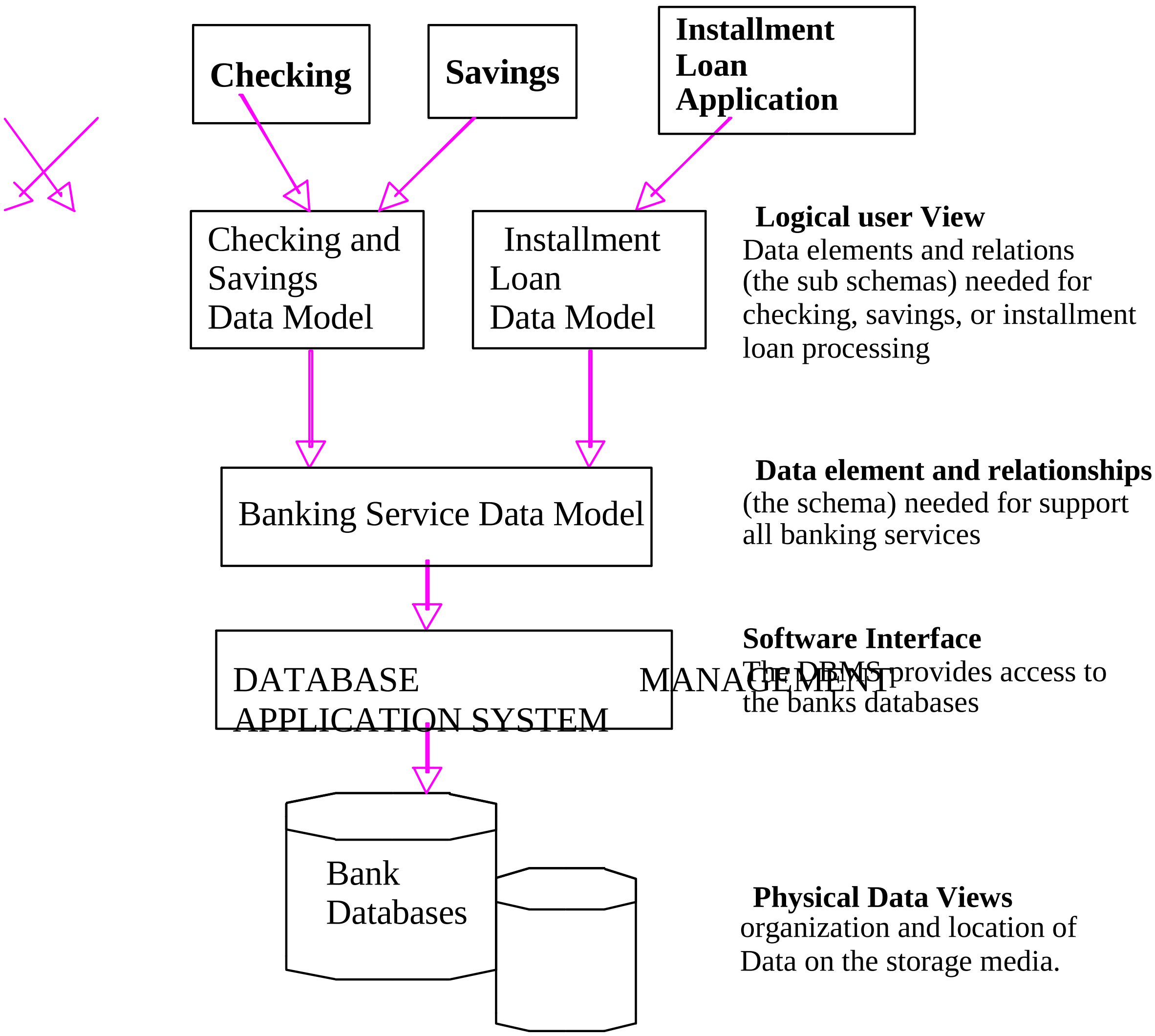


Figure 1: Database Development Structure

Note: Database development involves data planning and database design activities. Data models that support business process are used to develop databases that meet the information needs of users.

Figure 2: Examples of the logical and physical database views and the software interface of a banking service information system.



Although the normal forms are often defined informally in terms of the characteristics of tables, rigorous definitions of the normal forms are concerned with the characteristics of mathematical constructs known as relations. Whenever information is represented relationally, it is meaningful to consider the extent to which the representation is normalized.

Problems addressed by normalization

An Update Anomaly. Employee 519 is shown as having different addresses on different records.

An Insertion Anomaly. Until the new faculty member is assigned to teach at least one course, his details cannot be recorded.

A Deletion Anomaly. All information about Dr. Giddens is lost when he temporarily ceases to be assigned to any courses.

A table that is not sufficiently normalized can suffer from logical inconsistencies of various types, and from anomalies involving data operations. In such a table:

- The same information can be expressed on multiple records; therefore updates to the table may result in logical inconsistencies. For example, each record in an "Employees' Skills" table might contain an Employee ID, Employee Address, and Skill; thus a change of address for a particular employee will potentially need to be applied to multiple records (one for each of his skills). If the update is not carried through successfully—if, that is, the employee's address is updated on some records but not others—then the table is left in an inconsistent state. Specifically, the table provides conflicting answers to the question of what this particular employee's address is. This phenomenon is known as an update anomaly.

- There are circumstances in which certain facts cannot be recorded at all. For example, each record in a "Faculty and Their Courses" table might contain a Faculty ID, Faculty Name, Faculty Hire Date, and Course Code—thus we can record the details of any faculty member who teaches at least one course, but we cannot record the details of a newly-hired faculty member who has not yet been assigned to teach any courses. This phenomenon is known as an insertion anomaly.

- There are circumstances in which the deletion of data representing certain facts necessitates the deletion of data representing completely different facts. The "Faculty and Their Courses" table described in the previous example suffers from this type of anomaly, for if a faculty member temporarily ceases to be assigned to any courses, we must delete the last of the records on which that faculty member appears. This phenomenon is known as a deletion anomaly.

Ideally, a relational database table should be designed in such a way as to exclude the possibility of update, insertion, and deletion anomalies. The normal forms of relational database theory provide guidelines for deciding whether a particular design will be vulnerable to such anomalies. It is possible to correct an unnormalized design so as to make it adhere to the demands of the normal forms: this is called normalization. Removal of redundancies of the tables will lead to several tables, with referential integrity restrictions between them.

Normalization typically involves decomposing an unnormalized table into two or more tables that, were they to be combined (joined), would convey exactly the same information as the original table.

Background to normalization: definitions

- Functional Dependency: Attribute B has a functional dependency on attribute A i.e. A B if, for each value of attribute A, there is exactly one value of attribute B. If value of A is repeating in tuples then value of B will also repeat. In our example, Employee Address has a functional dependency on Employee ID, because a particular Employee ID value corresponds to one and only one Employee Address value. (Note that the reverse need not be true: several employees could live at the same address and therefore one Employee Address value could correspond to more than one Employee ID. Employee ID is therefore not functionally dependent on Employee Address.) An attribute may be functionally dependent either on a single attribute or on a combination of attributes. It is not possible to determine the extent to which a design is normalized without understanding what functional dependencies apply to the attributes within its tables; understanding this, in turn, requires knowledge of the problem domain. For example, an Employer may require certain employees to split their time between two locations, such as New York City and London, and therefore want to allow Employees to have more than one Employee Address. In this case, Employee Address would no longer be functionally dependent on Employee ID.

- Trivial Functional Dependency: A trivial functional dependency is a functional dependency of an attribute on a superset of itself. {Employee ID, Employee Address} {Employee Address} is trivial, as is {Employee Address} {Employee Address}.
- Full Functional Dependency: An attribute is fully functionally dependent on a set of attributes X if it is

- functionally dependent on X, and
- not functionally dependent on any proper subset of X. {Employee Address} has a functional dependency on {Employee ID, Skill}, but not a full functional dependency, because is also dependent on {Employee ID}.

- Transitive Dependency: A transitive dependency is an indirect functional dependency, one in which X Z only by virtue of *X Y and Y Z.*

- Multivalued Dependency: A multivalued dependency is a constraint according to which the presence of certain rows in a table implies the presence of certain other rows: see the Multivalued Dependency article for a rigorous definition.
- Join Dependency: A table T is subject to a join dependency if T can always be recreated by joining multiple tables each having a subset of the attributes of T.
- SuperKey: A superkey is an attribute or set of attributes that uniquely identifies rows within a table; in other words, two distinct rows are always guaranteed to have distinct superkeys. {Employee ID, Employee Address, Skill} would be a superkey for the "Employees' Skills" table; {Employee ID, Skill} would also be a superkey.
- Candidate Key: A candidate key is a minimal superkey, that is, a superkey for which we can say that no proper subset of it is also a superkey. {Employee Id, Skill} would be a candidate key for the "Employees' Skills" table.
- Non-Prime Attribute: A non-prime attribute is an attribute that does not occur in any candidate key. Employee Address would be a non-prime attribute in the "Employees' Skills" table.
- Primary Key: Most DBMSs require a table to be defined as having a single unique key, rather than a number of possible keys. A primary key is a key which the database designer has designated for this purpose.

3.2.2 History

Edgar F. Codd first proposed the process of normalization and what came to be known as the 1st normal form:

There is, in fact, a very simple elimination procedure which we shall call normalization. Through decomposition non-simple domains are replaced by "domains whose elements are atomic (non-decomposable) values."
—Edgar F. Codd, A Relational Model of Data for Large Shared Data Banks

In his paper, Edgar F. Codd used the term "non-simple" domains to describe a heterogeneous data structure, but later researchers would refer to such a structure as an abstract data type.

3.3 Normal Forms

The normal forms (abbrev. NF) of relational database theory provide criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies. The higher the normal form applicable to a table, the less vulnerable it is to inconsistencies and anomalies. Each table has a "highest normal form" (HNF): by definition, a table always meets the requirements of its HNF and of all normal forms lower than its HNF; also by definition, a table fails to meet the requirements of any normal form higher than its HNF.

First normal form: A table is in first normal form (1NF) if and only if it represents a relation. Given that database tables embody a relation-like form, the defining characteristic of one in first normal form is that it does not allow duplicate rows or nulls. Simply put, a table with a unique key (which, by definition, prevents duplicate rows) and without any nullable columns is in 1NF.

Second normal form: The criteria for second normal form (2NF) are:

- The table must be in 1NF.
- None of the non-prime attributes of the table are functionally dependent on a part (proper subset) of a candidate key; in other words, all functional dependencies of non-prime attributes on candidate keys are full functional dependencies. For example, consider an "Employees' Skills" table whose attributes are Employee ID, Employee Name, and Skill; and suppose that the combination of Employee ID and Skill uniquely identifies records within the table. Given that Employee Name depends on only one of those attributes – namely, Employee ID – the table is not in 2NF.
- In simple, a table is 2NF if it is in 1NF and all fields are dependant on the whole of the primary key, or a relation is in 2NF if it is in 1NF and every non-key attribute is fully dependent on each candidate key of the relation.
- Note that if none of a 1NF table's candidate keys are composite – i.e. every candidate key consists of just one attribute – then we can say immediately that the table is in 2NF.
- All columns must be a fact about the entire key, and not a subset of the key.

Third Normal Form: The criteria for third normal form (3NF) are:

- The table must be in 2NF.
- Transitive dependencies must be eliminated. All attributes must rely only on the primary key. So, if a database has a table with columns Student ID, Student, Company, and Company Phone Number, it is not in 3NF. This is because the Phone number relies on the Company. So, for it to be in 3NF, there must be a second table with Company and Company Phone Number columns; the Phone Number column in the first table would be removed.

Fourth normal form: A table is in fourth normal form (4NF) if and only if, for every one of its non-trivial multivalued dependencies $X \twoheadrightarrow Y$, *X is a superkey—that is, X is either a candidate key or a superset thereof.*

- For example, if you can have two phone numbers values and two email address values, then you should not have them in the same table.

Fifth normal form: The criteria for fifth normal form (5NF and also PJ/NF) are:

- The table must be in 4NF.
- There must be no non-trivial join dependencies that do not follow from the key constraints. A 4NF table is said to be in the 5NF if and only if every join dependency in it is implied by the candidate keys.

Domain/key Normal Form (or DKNF) requires that a table not be subject to any constraints other than domain constraints and key constraints.

Sixth Normal Form: According to the definition by Christopher J. Date and others, who extended database theory to take account of temporal and other interval data, a table is in sixth normal form (6NF) if and only if it satisfies no non-trivial (in the formal sense) join dependencies at all, , meaning that the fifth normal form is also satisfied. When referring to "join" in this context it should be noted that Date et al. additionally use generalized definitions of relational operators that also take account of interval data (e.g. from-date to-date) by conceptually breaking them down ("unpacking" them) into atomic units (e.g. individual days), with defined rules for joining interval data, for instance.

3.4 Denormalization

Databases intended for Online Transaction Processing (OLTP) are typically more normalized than databases intended for Online Analytical Processing (OLAP). OLTP Applications are characterized by a high volume of small transactions such as updating a sales record at a super market checkout counter. The expectation is that each transaction will leave the database in a consistent state. By contrast, databases intended for OLAP operations are primarily "read mostly" databases. OLAP applications tend to extract historical data that has accumulated over a long period of time. For such databases, redundant or "denormalized" data may facilitate Business Intelligence applications. Specifically, dimensional tables in a star schema often contain denormalized data. The denormalized or redundant data must be carefully controlled during

ETL processing, and users should not be permitted to see the data until it is in a consistent state. The normalized alternative to the star schema is the snowflake schema. It has never been proven that this denormalization itself provides any increase in performance, or if the concurrent removal of data constraints is what increases the performance. In many cases, the need for denormalization has waned as computers and RDBMS software have become more powerful, but since data volumes have generally increased along with hardware and software performance, OLAP databases often still use denormalized schemas.

Denormalization is also used to improve performance on smaller computers as in computerized cash-registers and mobile devices, since these may use the data for look-up only (e.g. price lookups). Denormalization may also be used when no RDBMS exists for a platform (such as Palm), or no changes are to be made to the data and a swift response is crucial.

3.5 Non-first normal form (NF^2 or N1NF)

In recognition that denormalization can be deliberate and useful, the non-first normal form is a definition of database designs which do not conform to the first normal form, by allowing "sets and sets of sets to be attribute domains" (Schek 1982). This extension is a (non-optimal) way of implementing hierarchies in relations. Some academics have dubbed this practitioner developed method, "First Ab-normal Form", Codd defined a relational database as using relations, so any table not in 1NF could not be considered to be relational.

Consider the following table:

Non-First Normal Form

Person	**Favorite Colors**
Bob	blue, red
Jane	green, yellow, red

Assume a person has several favorite colors. Obviously, favorite colors consist of a set of colors modeled by the given table.

To transform this NF^2 table into a 1NF an "unnest" operator is required which extends the relational algebra of the higher normal forms. The reverse operator is called "nest" which is not always the mathematical inverse of "unnest", although "unnest" is the mathematical inverse to "nest". Another constraint required is for the operators to be bijective, which is covered by the Partitioned Normal Form (PNF).

4.0 CONCLUSION

In the design and development of database management systems, organizations may use one kind of DBMS for daily transactions, and then move the detail unto another computer that uses another DBMS better suited for inquiries and analysis. Overall systems design decisions are performed by database administrators. The three most common organizations are hierarchical, network and relational models. A DBMS may provide one, two or all three models in designing database management systems.

5.0 SUMMARY

- Database design is the process of deciding how to organize data into records types and how the record types will relate to each other
- Database development may start with a top-down data planning process. Database administrators and designers work with corporate and end user management to develop an enterprise model that defines the basic business process of the enterprise
- Database normalization, sometimes referred to as canonical *synthesis, is a technique for designing relational database tables* to minimize duplication of information and, in so doing, to safeguard the database against certain types of logical or structural problems, namely data anomalies
- Edgar F. Codd first proposed the process of normalization and what came to be known as the 1st normal form:
- The normal forms (abbrev. NF) of relational database theory provide criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies.
- Databases intended for Online Transaction Processing (OLTP) are typically more normalized than databases intended for Online Analytical Processing (OLAP). OLTP Applications are characterized by a high volume of small transactions such as updating a sales record at a super market checkout counter.
- In recognition that denormalization can be deliberate and useful, the non-first normal form is a definition of database designs which do not conform to the first normal form, by allowing "sets and sets of sets to be attribute domains"

6.0 TUTOR-MARKED ASSIGNMENT

1. Mention the 5 phases in the development of database.
2. Identify the criteria for the second normal form (2NF).

7.0 REFERENCES/FURTHER READINGS

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

Date, C.J. "What First Normal Form Really Means" in Date on *Database: Writings 2000-2006 (Springer-Verlag, 2006), p. 128.*

Codd, E.F. "Is Your DBMS Really Relational?" Computerworld, October 14, 1985.

Coles, M. Sic Semper Null. 2007. SQL Server Central. Redgate Software.

Kent, William. "A Simple Guide to Five Normal Forms in Relational Database Theory", Communications of the ACM 26 (2), Feb. 1983, pp. 120-125.

Codd, E.F. "Further Normalization of the Data Base Relational Model." (Presented at Courant Computer Science Symposia Series 6, "Data Base Systems," New York City, May 24th-25th, 1971.) IBM Research Report RJ909 (August 31st, 1971). Republished in Randall J. Rustin (ed.), Data Base Systems: *Courant Computer Science Symposia Series 6. Prentice-Hall,* 1972.

Codd, E. F. "Recent Investigations into Relational Data Base Systems." IBM Research Report RJ1385 (April 23rd, 1974). Republished in *Proc. 1974 Congress (Stockholm, Sweden, 1974). New York,* N.Y.: North-Holland (1974).

Fagin, Ronald (September 1977). "Multivalued Dependencies and a New Normal Form for Relational Databases". ACM Transactions *on Database Systems 2 (1): 267. doi:10.1145/320557.320571.*

Date, Chris J.; Hugh Darwen, Nikos A. Lorentzos [January 2003]. "Chapter 10 Database Design, Section 10.4: Sixth Normal Form", *Temporal Data and the Relational Model: A Detailed Investigation into the Application of Interval and Relation Theory to the Problem of Temporal Database Management.* Oxford: Elsevier LTD, p176. ISBN 1558608559

O'Brien A. James, (2003). (11th Edition). Introduction to Information Systems, McGrw-Hill.

Zimyani, E. (June 2006). "Temporal Aggregates and Temporal Universal Quantification in Standard SQL". ACM *SIGMOD Record, volume 35, number 2. ACM.*

UNIT 2 STRUCTURED QUERY LANGUAGE (SQL)

CONTENTS

1.0 INTRODUCTION

SQL (Structured Query Language) is a database computer language designed for the retrieval and management of data in relational database management systems (RDBMS), database schema creation and modification, and database object access control management.

SQL is a standard interactive and programming language for querying and modifying data and managing databases. Although SQL is both an ANSI and an ISO standard, many database products support SQL with proprietary extensions to the standard language. The core of SQL is formed by a command language that allows the retrieval, insertion, updating, and deletion of data, and performing management and administrative functions. SQL also includes a Call Level Interface (SQL/CLI) for accessing and managing data and databases remotely.

The first version of SQL was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s. This version, initially called SEQUEL, was designed to manipulate and retrieve data stored in IBM's original relational database product, System R. The SQL language was later formally standardized by the American National Standards Institute (ANSI) in 1986. Subsequent versions of the SQL standard have been released as International Organization for Standardization (ISO) standards.

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

Common criticisms of SQL include a perceived lack of cross-platform portability between vendors, inappropriate handling of missing data (see *Null (SQL), and unnecessarily complex and occasionally ambiguous* language grammar and semantics.

SQL	
Paradigm	**Multi-paradigm**
Appeared in	**1974**
Designed by	**Donald D. Chamberlin** and Raymond F. Boyce
Developer	**IBM**
Latest release	**SQL:2006/ 2006**
Typing discipline	**static, strong**
Major implementations	Many
Dialects	**SQL-86, SQL-89,** SQL-92, SQL:1999, SQL: 2003, SQL:2006
Influenced by	**Datalog**
Influenced	**CQL, LINQ, Windows** PowerShell
OS	Cross-platform

2.0 OBJECTIVES

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

- define structure query language (SQL)
- trace the history and development process of SQL
- know the scope and extension of SQL
- identify the vital indices of SQL
- know what are the language elements
- know some of the criticism of SQL
- answer the question of alternatives to SQL

3.0 MAIN CONTENT

3.1 History

During the 1970s, a group at IBM San Jose Research Laboratory developed the System R relational database management system, based on the model introduced by Edgar F. Codd in his influential paper, A **Relational Model of Data for Large Shared Data Banks. Donald D.** Chamberlin and Raymond F. Boyce of IBM subsequently created the **Structured English Query Language (SEQUEL) to manipulate and** manage data stored in System R. The acronym SEQUEL was later changed to SQL because "SEQUEL" was a trademark of the UK-based Hawker Siddeley aircraft company.

The first non-commercial non-SQL RDBMS, Ingres, was developed in 1974 at the U.C. Berkeley. Ingres implemented a query language known as QUEL, which was later supplanted in the marketplace by SQL.

In the late 1970s, Relational Software, Inc. (now Oracle Corporation) saw the potential of the concepts described by Codd, Chamberlin, and Boyce and developed their own SQL-based RDBMS with aspirations of selling it to the U.S. Navy, CIA, and other government agencies. In the summer of 1979, Relational Software, Inc. introduced the first commercially available implementation of SQL, Oracle V2 (Version2) for VAX computers. Oracle V2 beat IBM's release of the System/38 RDBMS to market by a few weeks.

After testing SQL at customer test sites to determine the usefulness and practicality of the system, IBM began developing commercial products based on their System R prototype including System/38, SQL/DS, and DB2, which were commercially available in 1979, 1981, and 1983, respectively.

3.2 Standardization

SQL was adopted as a standard by ANSI in 1986 and ISO in 1987. In the original SQL standard, ANSI declared that the official pronunciation for SQL is "es queue el". However, many English-speaking database professionals still use the nonstandard pronunciation /'si kw l/ (like the word "sequel"). SEQUEL was an earlier IBM database language, a predecessor to the SQL language.

Until 1996, the National Institute of Standards and Technology (NIST) data management standards program was tasked with certifying SQL DBMS compliance with the SQL standard. In 1996, however, the NIST data management standards program was dissolved, and vendors are now relied upon to self-certify their products for compliance.

The SQL standard has gone through a number of revisions, as shown below:

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First published by ANSI. Ratified by ISO in 1987.
1989	SQL-89	FIPS 127-1	Minor revision, adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), Entry Level SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries, triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features.
2003	SQL:2003		Introduced XML-related features, window *functions, standardized sequences, and* columns with auto-generated values (including identity- columns).
2006	SQL:2006		ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it provides facilities that permit applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.

The SQL standard is not freely available. SQL: 2003 and SQL: 2006 may be purchased from ISO or ANSI. A late draft of SQL: 2003 is freely available as a zip archive, however, from Whitemarsh Information Systems Corporation. The zip archive contains a number of PDF files that define the parts of the SQL: 2003 specification.

3.3 Scope and Extensions

Procedural Extensions

SQL is designed for a specific purpose: to query data contained in a relational database. SQL is a set-based, declarative query language, not an imperative language such as C or BASIC. However, there are extensions to Standard SQL which add procedural programming language functionality, such as control-of-flow constructs. These are:

Source	Common Name	Full Name
ANSI/ISO Standard	SQL/PSM	SQL/Persistent Stored Modules
IBM	SQL PL	SQL Procedural Language (implements SQL/PSM)
Microsoft/ Sybase	T-SQL	Transact-SQL
MySQL	SQL/PSM	SQL/Persistent Stored Module (as in ISO SQL:2003)
Oracle	PL/SL	Procedural Language/SQL (based on Ada)
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL Structured Query Language (based on Oracle PL/SQL)
PostgreSQL	PL/PSM	Procedural Language/Persistent Stored Modules (implements SQL/PSM)

In addition to the standard SQL/PSM extensions and proprietary SQL extensions, procedural and object-oriented programmability is available on many SQL platforms via DBMS integration with other languages. The SQL standard defines SQL/JRT extensions (SQL Routines and Types for the Java Programming Language) to support Java code in SQL databases. SQL Server 2005 uses the SQLCLR (SQL Server Common Language Runtime) to host managed .NET assemblies in the database, while prior versions of SQL Server were restricted to using unmanaged extended stored procedures which were primarily written in C. Other database platforms, like MySQL and Postgres, allow functions to be written in a wide variety of languages including Perl, Python, Tcl, and C.

Additional Extensions

SQL: 2003 also defines several additional extensions to the standard to increase SQL functionality overall. These extensions include:

The SQL/CLI, or Call-Level Interface, extension is defined in ISO/IEC 9075-3:2003. This extension defines common interfacing components (structures and procedures) that can be used to execute SQL statements from applications written in other programming languages. The SQL/CLI extension is defined in such a way that SQL statements and SQL/CLI procedure calls are treated as separate from the calling application's source code.

The SQL/MED, or Management of External Data, extension is defined by ISO/IEC 9075-9:2003. SQL/MED provides extensions to SQL that define foreign-data wrappers and datalink types to allow SQL to manage external data. External data is data that is accessible to, but not managed by, an SQL-based DBMS.

The SQL/OLB, or Object Language Bindings, extension is defined by ISO/IEC 9075-10:2003. SQL/OLB defines the syntax and symantics of SQLJ, which is SQL embedded in Java. The standard also describes mechanisms to ensure binary portability of SQLJ applications, and specifies various Java packages and their contained classes.
The SQL/Schemata, or Information and Definition Schemas, extension is defined by ISO/IEC 9075-11:2003. SQL/Schemata defines the Information Schema and Definition Schema, providing a common set of tools to make SQL databases and objects self-describing. These tools include the SQL object identifier, structure and integrity constraints, security and authorization specifications, features and packages of ISO/IEC 9075, support of features provided by SQL-based DBMS implementations, SQL-based DBMS implementation information and sizing items, and the values supported by the DBMS implementations.

The SQL/JRT, or SQL Routines and Types for the Java **Programming Language, extension is defined by ISO/IEC** 9075-13:2003. SQL/JRT specifies the ability to invoke static Java methods as routines from within SQL applications. It also calls for the ability to use Java classes as SQL structured user-defined types.

The SQL/XML, or XML-Related Specifications, extension is defined by ISO/IEC 9075-14:2003. SQL/XML specifies SQL-based extensions for using XML in conjunction with SQL. The XML data type is introduced, as well as several routines, functions, and XML-to-SQL data type mappings to support manipulation and storage of XML in an SQL database.

The SQL/PSM, or Persistent Stored Modules, extension is defined by ISO/IEC 9075-4:2003. SQL/PSM standardizes procedural extensions for SQL, including flow of control, condition handling, statement condition signals and resignals, cursors and local variables, and assignment of expressions to variables and parameters. In addition, SQL/PSM formalizes declaration and maintenance of persistent database language routines (e.g., "stored procedures").

3.4 Language Elements

This chart shows several of the SQL language elements that compose a single statement.

The SQL language is sub-divided into several language elements, including:

- Statements which may have a persistent effect on schemas and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
- Queries which retrieve data based on specific criteria.
- Expressions which can produce either scalar values or tables consisting of columns and rows of data.
- Predicates which specify conditions that can be evaluated to SQL three-valued logic (3VL) Boolean truth values and which are used to limit the effects of statements and queries, or to change program flow.
- Clauses, which are in some cases optional, constituent components of statements and queries.
- Whitespace is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.
- SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.

Queries

The most common operation in SQL databases is the query, which is performed with the declarative SELECT keyword. SELECT retrieves data from a specified table, or multiple related tables, in a database. While often grouped with Data Manipulation Language (DML) statements, the standard SELECT query is considered separate from SQL DML, as it has no persistent effects on the data stored in a database. Note that there are some platform-specific variations of SELECT that can persist their effects in a database, such as the SELECT INTO syntax that exists in some databases.

SQL queries allow the user to specify a description of the desired result set, but it is left to the devices of the database management system (DBMS) to plan, optimize, and perform the physical operations

necessary to produce that result set in as efficient a manner as possible.
An SQL query includes a list of columns to be included in the final result immediately following the SELECT keyword. An asterisk ("*") can also be used as a "wildcard" indicator to specify that all available columns of a table (or multiple tables) are to be returned. SELECT is the most complex statement in SQL, with several optional keywords and clauses, including:

- The FROM clause which indicates the source table or tables from which the data is to be retrieved. The FROM clause can include optional JOIN clauses to join related tables to one another based on user-specified criteria.

- The WHERE clause includes a comparison predicate, which is used to restrict the number of rows returned by the query. The WHERE clause is applied before the GROUP BY clause. The WHERE clause eliminates all rows from the result set where the comparison predicate does not evaluate to True.

- The GROUP BY clause is used to combine, or group, rows with related values into elements of a smaller set of rows. GROUP BY is often used in conjunction with SQL aggregate functions or to eliminate duplicate rows from a result set.

- The HAVING clause includes a comparison predicate used to eliminate rows after the GROUP BY clause is applied to the result set. Because it acts on the results of the GROUP BY clause, aggregate functions can be used in the HAVING clause predicate.

- The ORDER BY clause is used to identify which columns are used to sort the resulting data, and in which order they should be sorted (options are ascending or descending). The order of rows returned by an SQL query is never guaranteed unless an ORDER BY clause is specified.

Data Definition

The second group of keywords is the Data Definition Language (DDL). DDL allows the user to define new tables and associated elements. Most commercial SQL databases have proprietary extensions in their DDL, which allow control over nonstandard features of the database system.
The most basic items of DDL are the CREATE, ALTER, RENAME, TRUNCATE and DROP statements:

- CREATE causes an object (a table, for example) to be created within the database.
- DROP causes an existing object within the database to be deleted, usually irretrievably.
- TRUNCATE deletes all data from a table (non-standard, but common SQL statement).
- ALTER statement permits the user to modify an existing object in various ways -- for example, adding a column to an existing table.

Data Control

The third group of SQL keywords is the Data Control Language (DCL). DCL handles the authorization aspects of data and permits the user to control who has access to see or manipulate data within the database. Its two main keywords are:

- GRANT authorizes one or more users to perform an operation or a set of operations on an object.
- REVOKE removes or restricts the capability of a user to perform an operation or a set of operations.

3.5 Criticisms of SQL

Technically, SQL is a declarative computer language for use with "SQL databases". Theorists and some practitioners note that many of the original SQL features were inspired by, but violated, the relational model for database management and its tuple calculus realization. Recent extensions to SQL achieved relational completeness, but have worsened the violations, as documented in The Third Manifesto.
In addition, there are also some criticisms about the practical use of SQL:

- Implementations are inconsistent and, usually, incompatible between vendors. In particular date and time syntax, string concatenation, nulls, and comparison case sensitivity often vary from vendor to vendor.
- The language makes it too easy to do a Cartesian join (joining all possible combinations), which results in "run-away" result sets when WHERE clauses are mistyped. Cartesian joins are so rarely used in practice that requiring an explicit CARTESIAN keyword may be warranted.

SQL 1992 introduced the CROSS JOIN keyword that allows the user to make clear that a cartesian join is intended, but the shorthand "comma-join" with no predicate is still acceptable syntax.

- It is also possible to misconstruct a WHERE on an update or delete, thereby affecting more rows in a table than desired.

- The grammar of SQL is perhaps unnecessarily complex, borrowing a COBOL-like keyword approach, when a function-influenced syntax could result in more re-use of fewer grammar and syntax rules. This is perhaps due to IBM's early goal of making the language more English-like so that it is more approachable to those without a mathematical or programming background. (Predecessors to SQL were more mathematical.)

Reasons for lack of portability

Popular implementations of SQL commonly omit support for basic features of Standard SQL, such as the DATE or TIME data types, preferring variations of their own. As a result, SQL code can rarely be ported between database systems without modifications.

There are several reasons for this lack of portability between database systems:

- The complexity and size of the SQL standard means that most databases do not implement the entire standard.

- The standard does not specify database behavior in several important areas (e.g. indexes, file storage...), leaving it up to implementations of the database to decide how to behave.

- The SQL standard precisely specifies the syntax that a conforming database system must implement. However, the standard's specification of the semantics of language constructs is less well-defined, leading to areas of ambiguity.

- Many database vendors have large existing customer bases; where the SQL standard conflicts with the prior behavior of the vendor's database, the vendor may be unwilling to break backward compatibility.

3.6 Alternatives to SQL

A distinction should be made between alternatives to relational query languages and alternatives to SQL. The lists below are proposed alternatives to SQL, but are still (nominally) relational. See navigational database for alternatives to relational:

- IBM Business System 12 (IBM BS12)

- Tutorial D
- Hibernate Query Language (HQL) - A Java-based tool that uses modified SQL
- Quel introduced in 1974 by the U.C. Berkeley Ingres project.
- Object Query Language
- Datalog
- .QL - object-oriented Datalog
- LINQ
- QLC - Query Interface to Mnesia, ETS, Dets, etc (Erlang programming language)
- 4D Query Language (4D QL)
- QBE (Query By Example) created by Moshè Zloof, IBM 1977
- Aldat Relational Algebra and Domain algebra

4.0 CONCLUSION

The structured query language (SQL) has become the official dominant language for writing database management system. This language differs from conventional methods of computer language writing, because it is not necessarily procedural. An SQL statement is not really a command to computer but it is rather a description of some of the daatcotained in a database. SQL is not procedural because it does not give step-by-step commands to the computer or database. It describes data and sometimes instructs the database to do something with the data. Irrespective of this, SQL has it own criticism.

5.0 SUMMARY

- SQL (Structured Query Language) is a database computer language designed for the retrieval and management of data in relational database management systems (RDBMS), database schema creation and modification, and database object access control management.

- During the 1970s, a group at IBM San Jose Research Laboratory developed the System R relational DATABASE MANAGEMENT APPLICATION SYSTEM based on the model, introduced by Edgar F. Codd in his influential paper, A Relational Model of Data for Large **Shared Data Banks.**

- SQL was adopted as a standard by ANSI in 1986 and ISO in 1987. In the original SQL standard, ANSI declared that the official pronunciation for SQL is "es queue el".

- SQL is designed for a specific purpose: to query data contained in a relational database. SQL is a set-based, declarative query language, not an imperative language such as C or BASIC.

- This chart shows several of the SQL language elements that compose a single statement.

- Technically, SQL is a declarative computer language for use with "SQL databases". Theorists and some practitioners note that many of the original SQL features were inspired by, but violated, the relational model for database management and its tuple calculus realization.

- A distinction should be made between alternatives to relational query languages and alternatives to SQL

6.0 TUTOR-MARKED ASSIGNMENT

List and discuss the sub-divisions of the language of structures query language

7.0 REFERENCES/FURTHER READINGS

Chapple, Mike. "SQL Fundamentals (HTML). About.com: Databases. About.com.

"Structured Query Language (SQL)" (HTML). International Business Machines (October 27, 2006).

Codd, E.F. (June 1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (No. 6): pp. 377–387. Association for Computing Machinery. doi: 10.1145/362384.362685.

Chamberlin, Donald D.; Boyce, Raymond F. (1974). "SEQUEL: A Structured English Query Language". Proceedings of the 1974 *ACM SIGFIDET Workshop on Data Description, Access and Control: pp. 249–264. Association for Computing Machinery.*

a b Oppel, Andy (March 1, 2004). Databases Demystified. San Francisco,

CA: McGraw-Hill Osborne Media, pp. 90-91. ISBN 0-07-225364-9.

"History of IBM, 1978 (HTML). IBM Archives. IBM.

Chapple, Mike (?). "SQL Fundamentals" (HTML). About.com. About.com, A New York Times Company. Retrieved on 2007-08-30.

Melton, Jim; Alan R Simon (1993). Understanding the New SQL: A *Complete Guide. Morgan Kaufmann, 536. ISBN:* 1558602453. “chapter 1.2 What is SQL? SQL (correctly pronounced "ess cue ell," instead of the somewhat common "sequel"), is a...”

"Understand SQL". www.faqs.org/docs/.

Doll, Shelley (June 19, 2002). "Is SQL a Standard Anymore?" (HTML). *TechRepublic's Builder.com. TechRepublic. Retrieved on* 2007-06-09.

ISO/IEC 9075-11:2003: Information and Definition Schemas (SQL/Schemata), 2003, pp. p. 1.

ANSI/ISO/IEC International Standard (IS). Database Language SQL—Part 2: Foundation (SQL/Foundation). 1999.

"INTO Clause (Transact-SQL)" (HTML). SQL Server 2005 Books *Online. Microsoft (2007). Retrieved on 2007-06-17.*

M. Negri, G. Pelagatti, L. Sbattella (1989) Semantics and problems of *universal quantification in SQL.*

Claudio Fratarcangeli (1991) Technique for universal quantification in *SQL.*

Jalal Kawash Complex quantification in Structured Query Language *(SQL): a Tutorial Using Relational Calculus - Journal of* Computers in Mathematics and Science Teaching ISSN 0731-9258 Volume 23, Issue 2, 2004 AACE Norfolk, VA.

UNIT 3 DATABASE AND INFORMATION SYSTEMS SECURITY

CONTENTS

1.0 INTRODUCTION

Data security is the means of ensuring that data is kept safe from corruption and that access to it is suitably controlled. Thus data security helps to ensure privacy. It also helps in protecting personal data.

Information security means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction. The terms information security, computer security and information assurance are frequently used interchangeably. These fields are interrelated and share the common goals of protecting the confidentiality, integrity and availability of information; however, there are some subtle differences between them. These differences lie primarily in the approach to the subject, the methodologies used, and the areas of concentration. Information security is concerned with the confidentiality, integrity and availability of data regardless of the form the data may take: electronic, print, or other forms.

Governments, military, financial institutions, hospitals, and private businesses amass a great deal of confidential information about their employees, customers, products, research, and financial status. Most of this information is now collected, processed and stored on electronic computers and transmitted across networks to other computers. Should confidential information about a businesses customers or finances or new product line fall into the hands of a competitor, such a breach of security could lead to lost business, law suits or even bankruptcy of the business. Protecting confidential information is a business requirement, and in many cases also an ethical and legal requirement. For the individual, information security has a significant effect on privacy, which is viewed very differently in different cultures.

The field of information security has grown and evolved significantly in recent years. As a career choice there are many ways of gaining entry into the field. It offers many areas for specialization including Information Systems Auditing, Business Continuity Planning and Digital Forensics Science, to name a few.

2.0 OBJECTIVES

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

- understand the concepts of the CIA Trade in respect of information systems security
- know the components of the Donn Parker model for the classic Triad
- identify the different types of information access control and how they differ from each other
- differentiate Discretionary and Mandatory Access Control Policies
- know the Proposed OODBMS Security Models
- differentiate between the OODBMS models
- defining appropriate procedures and protection requirements for information security
- define cryptography and know its applications in data security.

3.0 MAIN CONTENT

3.1 Basic Principles

3.1.1 Key Concepts

For over twenty years information security has held that confidentiality, integrity and availability (known as the CIA Triad) are the core principles of information system security.

Confidentiality

Confidentiality is the property of preventing disclosure of information to unauthorized individuals or systems. For example, a credit card transaction on the Internet requires the credit card number to be transmitted from the buyer to the merchant and from the merchant to a transaction processing network. The system attempts to enforce confidentiality by encrypting the card number during transmission, by limiting the places where it might appear (in databases, log files, backups, printed receipts, and so on), and by restricting access to the places where it is stored. If an unauthorized party obtains the card number in any way, a breach of confidentiality has occurred.

Breaches of confidentiality take many forms. Permitting someone to look over your shoulder at your computer screen while you have confidential data displayed on it could be a breach of confidentiality. If a laptop computer containing sensitive information about a company's employees is stolen or sold, it could result in a breach of confidentiality. Giving out confidential information over the telephone is a breach of confidentiality if the caller is not authorized to have the information.

Confidentiality is necessary (but not sufficient) for maintaining the privacy of the people whose personal information a system holds.

Integrity

In information security, integrity means that data cannot be modified without authorization. (This is not the same thing as referential integrity in databases.) Integrity is violated when an employee (accidentally or with malicious intent) deletes important data files, when a computer virus infects a computer, when an employee is able to modify his own salary in a payroll database, when an unauthorized user vandalizes a web site, when someone is able to cast a very large number of votes in an online poll, and so on.

Availability

For any information system to serve its purpose, the information must be available when it is needed. This means that the computing systems used to store and process the information, the security controls used to protect it, and the communication channels used to access it must be functioning correctly. High availability systems aim to remain available at all times, preventing service disruptions due to power outages, hardware failures, and system upgrades. Ensuring availability also involves preventing denial-of-service attacks.

In 2002, Donn Parker proposed an alternative model for the classic CIA triad that he called the six atomic elements of information. The elements are confidentiality, possession, integrity, authenticity, availability, and utility. The merits of the Parkerian hexad are a subject of debate amongst security professionals.

3.1.2 Authenticity

In computing, e-Business and information security it is necessary to ensure that the data, transactions, communications or documents (electronic or physical) are genuine (i.e. they have not been forged or fabricated.)

3.1.3 Non-Repudiation

In law, non-repudiation implies ones intention to fulfill their obligations to a contract. It also implies that one party of a transaction can not deny having received a transaction nor can the other party deny having sent a transaction.

Electronic commerce uses technology such as digital signatures and encryption to establish authenticity and non-repudiation.

3.1.4 Risk Management

Security is everyone’s responsibility. Security awareness poster. U.S. Department of Commerce/Office of Security.

A comprehensive treatment of the topic of risk management is beyond the scope of this article. We will however, provide a useful definition of risk management, outline a commonly used process for risk management, and define some basic terminology.

The CISA Review Manual 2006 provides the following definition of risk management: "Risk management is the process of identifying *vulnerabilities and threats to the information resources used by an organization in achieving business objectives, and deciding what countermeasures, if any, to take in reducing risk to an acceptable level, based on the value of the information resource to the organization."*

There are two things in this definition that may need some clarification. First, the process of risk management is an ongoing iterative process. It must be repeated indefinitely. The business environment is constantly changing and new threats and vulnerabilities emerge every day. Second, the choice of countermeasures (controls) used to manage risks must strike a balance between productivity, cost, effectiveness of the countermeasure, and the value of the informational asset being protected.

Risk is the likelihood that something bad will happen that causes harm to an informational asset (or the loss of the asset). A vulnerability is a weakness that could be used to endanger or cause harm to an informational asset. A threat is anything (man made or act of nature) that has the potential to cause harm.

The likelihood that a threat will use a vulnerability to cause harm creates a risk. When a threat does use a vulnerability to inflict harm, it has an impact. In the context of information security, the impact is a loss of availability, integrity, and confidentiality, and possibly other losses (lost income, loss of life, loss of real property). It should be pointed out that it is not possible to identify all risks, nor is it possible to eliminate all risk. The remaining risk is called residual risk.

A risk assessment is carried out by a team of people who have knowledge of specific areas of the business. Membership of the team may vary over time as different parts of the business are assessed. The assessment may use a subjective qualitative analysis based on informed opinion, or where reliable dollar figures and historical information is available, the analysis may use quantitative analysis.

3.1.5 Controls

When Management chooses to mitigate a risk, they will do so by implementing one or more of three different types of controls.

Administrative

Administrative controls (also called procedural controls) consist of approved written policies, procedures, standards and guidelines. Administrative controls form the framework for running the business and managing people. They inform people on how the business is to be run and how day to day operations are to be conducted. Laws and regulations created by government bodies are also a type of administrative control because they inform the business. Some industry sectors have policies, procedures, standards and guidelines that must be followed - the Payment Card Industry (PCI) Data Security Standard required by Visa and Master Card is such an example. Other examples of administrative controls include the corporate security policy, password policy, hiring policies, and disciplinary policies.

Administrative controls form the basis for the selection and implementation of logical and physical controls. Logical and physical controls are manifestations of administrative controls. Administrative controls are of paramount importance.

Logical

Logical controls (also called technical controls) use software and data to monitor and control access to information and computing systems. For example: passwords, network and host based firewalls, network intrusion detection systems, access control lists, and data encryption are logical controls.

An important logical control that is frequently overlooked is the **principle of least privilege. The principle of least privilege requires that** an individual, program or system process is not granted any more access privileges than are necessary to perform the task. A blatant example of the failure to adhere to the principle of least privilege is logging into Windows as user Administrator to read Email and surf the Web. Violations of this principle can also occur when an individual collects additional access privileges over time. This happens when employees' job duties change, or they are promoted to a new position, or they transfer to another department. The access privileges required by their new duties are frequently added onto their already existing access privileges which may no longer be necessary or appropriate.

Physical

Physical controls monitor and control the environment of the work place and computing facilities. They also monitor and control access to and from such facilities. For example: doors, locks, heating and air conditioning, smoke and fire alarms, fire suppression systems, cameras, barricades, fencing, security guards, cable locks, etc. Separating the network and work place into functional areas are also physical controls.

An important physical control that is frequently overlooked is the **separation of duties. Separation of duties ensures that an individual can** not complete a critical task by himself. For example: an employee who submits a request for reimbursement should not also be able to authorize payment or print the check. An applications programmer should not also be the server administrator or the database administrator - these roles and responsibilities must be separated from one another.

3.2 Database Security

Database security is primarily concerned with the secrecy of data. Secrecy means protecting a database from unauthorized access by users and software applications.

Secrecy, in the context of data base security, includes a variety of threats incurred through unauthorized access. These threats range from the intentional theft or destruction of data to the acquisition of information through more subtle measures, such as inference. There are three generally accepted categories of secrecy-related problems in data base systems:

1. **The improper release of information from reading data that was intentionally or accidentally accessed by unauthorized users. Securing data bases from unauthorized access is more**

difficult than controlling access to files managed by operating systems. This problem arises from the finer granularity that is used by databases when handling files, attributes, and values. This type of problem also includes the violations to secrecy that result from the problem of inference, which is the deduction of unauthorized information from the observation of authorized information. Inference is one of the most difficult factors to control in any attempts to secure data. Because the information in a database is semantically related, it is possible to determine the value of an attribute without accessing it directly. Inference problems are most serious in statistical databases where users can trace back information on individual entities from the statistical aggregated data.

2. **The Improper Modification of Data. This threat includes** violations of the security of data through mishandling and modifications by unauthorized users. These violations can result from errors, viruses, sabotage, or failures in the data that arise from access by unauthorized users.

3. **Denial-Of-Service Threats. Actions that could prevent users** from using system resources or accessing data are among the most serious. This threat has been demonstrated to a significant degree recently with the SYN flooding attacks against network service providers.

Discretionary vs. Mandatory Access Control Policies

Both traditional relational data base management system (RDBMS) security models and OO data base models make use of two general types of access control policies to protect the information in multilevel systems. The first of these policies is the discretionary policy. In the discretionary access control (DAC) policy, access is restricted based on the authorizations granted to the user.

The mandatory access control (MAC) policy secures information by assigning sensitivity levels, or labels, to data entities. MAC policies are generally more secure than DAC policies and they are used in systems in which security is critical, such as military applications. However, the price that is usually paid for this tightened security is reduced performance of the data base management system. Most MAC policies also incorporate DAC measures as well.

3.3 Relational DBMS Security

The principal methods of security in traditional RDBMSs are through the appropriate use and manipulation of views and the structured query language (SQL) GRANT and REVOKE statements. These measures are reasonably effective because of their mathematical foundation in relational algebra and relational calculus.

3.3.1 View-Based Access Control

Views allow the database to be conceptually divided into pieces in ways that allow sensitive data to be hidden from unauthorized users. In the relational model, views provide a powerful mechanism for specifying data-dependent authorizations for data retrieval.

Although the individual user who creates a view is the owner and is entitled to drop the view, he or she may not be authorized to execute all privileges on it. The authorizations that the owner may exercise depend on the view semantics and on the authorizations that the owner is allowed to implement on the tables directly accessed by the view. For the owner to exercise a specific authorization on a view that he or she creates, the owner must possess the same authorization on all tables that the view uses. The privileges the owner possesses on the view are determined at the time of view definition. Each privilege the owner possesses on the tables is defined for the view. If, later on, the owner receives additional privileges on the tables used by the view, these additional privileges will not be passed onto the view. In order to use the new privileges within a view, the owner will need to create a new view.

The biggest problem with view-based mandatory access controls is that it is impractical to verify that the software performs the view interpretation and processing. If the correct authorizations are to be assured, the system must contain some type of mechanism to verify the classification of the sensitivity of the information in the database. The classification must be done automatically, and the software that handles the classification must be trusted. However, any trusted software for the automatic classification process would be extremely complex. Furthermore, attempting to use a query language such as SQL to specify classifications quickly become convoluted and complex. Even when the complexity of the classification scheme is overcome, the view can do nothing more than limit what the user sees — it cannot restrict the operations that may be performed on the views.

3.4 Proposed OODBMS Security Models

Currently only a few models use discretionary access control measures in secure object-oriented data base management systems.

Explicit Authorizations

The ORION authorization model permits access to data on the basis of explicit authorizations provided to each group of users. These authorizations are classified as positive authorizations because they specifically allow a user access to an object. Similarly, a negative authorization is used to specifically deny a user access to an object.

The placement of an individual into one or more groups is based on the role that the individual plays in the organization. In addition to the positive authorizations that are provided to users within each group, there are a variety of implicit authorizations that may be granted based on the relationships between subjects and access modes.

Data-Hiding Model

A similar discretionary access control secure model is the data-hiding model proposed by Dr. Elisa Bertino of the Universita' di Genova. This model distinguishes between public methods and private methods.

The data-hiding model is based on authorizations for users to execute methods on objects. The authorizations specify which methods the user is authorized to invoke. Authorizations can only be granted to users on public methods. However, the fact that a user can access a method does not automatically mean that the user can execute all actions associated with the method. As a result, several access controls may need to be performed during the execution, and all of the authorizations for the different accesses must exist if the user is to complete the processing.

Similar to the use of GRANT statements in traditional relational data base management systems, the creator of an object is able to grant authorizations to the object to different users. The "creator" is also able to revoke the authorizations from users in a manner similar to REVOKE statements. However, unlike traditional RDBMS GRANT statements, the data-hiding model includes the notion of protection mode. When authorizations are provided to users in the protection mode, the authorizations actually checked by the system are those of the creator and not the individual executing the method. As a result, the creator is able to grant a user access to a method without granting the user the authorizations for the methods called by the original method. In other words, the creator can provide a user access to specific data without being forced to give the user complete access to all related information in the object.

3.5 Security Classification for Information

An important aspect of information security and risk management is recognizing the value of information and defining appropriate procedures and protection requirements for the information. Not all information is equal and so not all information requires the same degree of protection. This requires information to be assigned a security classification.

Some factors that influence which classification information should be assigned include how much value that information has to the organization, how old the information is and whether or not the information has become obsolete. Laws and other regulatory requirements are also important considerations when classifying information.

Common information security classification labels used by the business sector are: public, sensitive, private, confidential. Common information security classification labels used by government are: **Unclassified, Sensitive But Unclassified, Restricted, Confidential, Secret, Top Secret and their non-English equivalents.**

All employees in the organization, as well as business partners, must be trained on the classification schema and understand the required security controls and handling procedures for each classification. The classification a particular information asset has been assigned should be reviewed periodically to ensure the classification is still appropriate for the information and to ensure the security controls required by the classification are in place.

Access control:Access to protected information must be restricted to people who are authorized to access the information. The computer programs, and in many cases the computers that process the information, must also be authorized. This requires that mechanisms be in place to control the access to protected information. The sophistication of the access control mechanisms should be in parity with the value of the information being protected - the more sensitive or valuable the information the stronger the control mechanisms need to be. The foundation on which access control mechanisms are built start with identification and authentication.

Identification is an assertion of who someone is or what something is. If a person makes the statement "Hello, my name is John Doe." they are making a claim of who they are. However, their claim may or may not be true. Before John Doe can be granted access to protected information it will be necessary to verify that the person claiming to be John Doe really is John Doe.

Authentication is the act of verifying a claim of identity. When John Doe goes into a bank to make a withdrawal, he tells the bank teller he is John Doe (a claim of identity). The bank teller asks to see a photo ID, so he hands the teller his drivers' license. The bank teller checks the license to make sure it has John Doe printed on it and compares the photograph on the license against the person claiming to be John Doe. If the photo and name match the person, then the teller has authenticated that John Doe is who he claimed to be.

On computer systems in use today, the Username is the most common form of identification and the Password is the most common form of authentication. Usernames and passwords have served their purpose but in our modern world they are no longer adequate. Usernames and passwords are slowly being replaced with more sophisticated authentication mechanisms.

After a person, program or computer has successfully been identified and authenticated then it must be determined what informational resources they are permitted to access and what actions they will be allowed to perform (run, view, create, delete, or change). This is called **authorization.**

Authorization to access information and other computing services begins with administrative policies and procedures. The polices prescribe what information and computing services can be accessed, by whom, and under what conditions. The access control mechanisms are then configured to enforce these policies.

Different computing systems are equipped with different kinds of access control mechanisms, some may offer a choice of different access control mechanisms. The access control mechanism a system offers will be based upon one of three approaches to access control or it may be derived from a combination of the three approaches.

The non-discretionary approach consolidates all access control under a centralized administration. The access to information and other resources is usually based on the individuals function (role) in the organization or the tasks the individual must perform. The discretionary approach gives the creator or owner of the information resource the ability to control access to those resources. In the Mandatory access control approach, access is granted or denied bases upon the security classification assigned to the information resource.

3.6 Cryptography

Information security uses cryptography to transform usable information into a form that renders it unusable by anyone other than an authorized user; this process is called encryption. Information that has been encrypted (rendered unusable) can be transformed back into its original usable form by an authorized user, who possesses the cryptographic key, through the process of decryption. Cryptography is used in information security to protect information from unauthorized or accidental discloser while the information is in transit (either electronically or physically) and while information is in storage.

Cryptography provides information security with other useful applications as well including improved authentication methods, message digests, digital signatures, non-repudiation, and encrypted network communications.

Cryptography can introduce security problems when it is not implemented correctly. Cryptographic solutions need to be implemented using industry accepted solutions that have undergone rigorous peer review by independent experts in cryptography. The length and strength of the encryption key is also an important consideration. A key that is weak or too short will produce weak encryption. The keys used for encryption and decryption must be protected with the same degree of rigor as any other confidential information. They must be protected from unauthorized disclosure and destruction and they must be available when needed.

Process

The terms reasonable and prudent person, due care and due **diligence have been used in the fields of Finance, Securities, and Law** for many years. In recent years these terms have found their way into the fields of computing and information security. U.S.A. Federal Sentencing Guidelines now make it possible to hold corporate officers liable for failing to exercise due care and due diligence in the management of their information systems. In the business world, stockholders, customers, business partners and governments have the expectation that corporate officers will run the business in accordance with accepted business practices and in compliance with laws and other regulatory requirements. This is often described as the "reasonable and prudent person" rule.

3.7 Disaster Recovery Planning

• What is Disaster Recovery Planning

Disaster Recovery Planning is all about continuing an IT service.

You need 2 or more sites, one of them is primary, which is planned to be recovered. The alternate site may be online...meaning production data is simultaneously transferred to both sites (sometime called as HOT Sites), may be offline...meaning data is tranferred after a certain delay through other means, (sometimes called as a WARM site) or even may not be transferred at all, but may have a replica IT system of the original site, which will be started whenever the primary site faces a disaster (sometimes called a COLD site).

• How are DRP and BCP different

Though DRP is part of the BCP process, DRP focusses on IT systems recovery and BCP on the entire business.

• How are DRP and BCP related

DRP is one of the recovery activities during execution of a Business Continuity Plan.

4.0 CONCLUSION

Data and information systems security is the ongoing process of exercising due care and due diligence to protect information, and information systems, from unauthorized access, use, disclosure, destruction, modification, or disruption or distribution. The never **ending process of information security involves ongoing training,** assessment, protection, monitoring & detection, incident response & repair, documentation, and review.

5.0 SUMMARY

This unit can be summarized as follows:

Data security is the means of ensuring that data is kept safe from corruption and that access to it is suitably controlled

• Information Security means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction. The terms information security, computer security and information assurance are frequently used interchangeably.

- For over twenty years information security has held that confidentiality, integrity and availability (known as the CIA Triad) are the core principles of information system security.
- The principal methods of security in traditional RDBMSs are through the appropriate use and manipulation of views and the structured query language (SQL) GRANT and REVOKE statements.
- Authentication is the act of verifying a claim of identity.
- Currently only a few models use discretionary access control measures in secure object-oriented data base management systems.
- An important aspect of information security and risk management is recognizing the value of information and defining appropriate procedures and protection requirements for the information.
- Information security uses cryptography to transform usable information into a form that renders it unusable by anyone other than an authorized user; this process is called encryption.
- Disaster Recovery Planning is all about continuing an IT service. You need 2 or more sites, one of them is primary, which is planned to be recovered.

6.0 TUTOR-MARKED ASSIGNMENT

1. List Donn Parker's 6 atomic elements of CIA Triad of information security.
2. Briefly discuss Disaster Recovery Planning in the security of DBMS.

7.0 REFERENCES/FURTHER READINGS

44 U.S.C § 3542 (b) (1) (2006)

Blackwell Encyclopedia of Management Information System, Vol. III, Edited by Gordon B. Davis.

Harris, Shon (2003). All-in-one CISSP Certification Exam Guide, 2nd Ed., /dmirror/http/en.wikipedia.org/w/Emeryville, CA: McGraw-Hill/Osborne.

ISACA (2006). CISA Review Manual 2006. Information Systems Audit and Control Association, p. 85. ISBN 1-933284-15-3.

Quist, Arvin S. (2002). "Security Classification of Information (HTML). Volume 1. Introduction, History, and Adverse Impacts. Oak Ridge Classification Associates, LLC. Retrieved on 2007-01-11.

UNIT 4 DATABASE ADMINISTRATOR AND ADMINISTRATION

CONTENTS

1.0 INTRODUCTION

A database administrator (DBA) is a person who is responsible for the environmental aspects of a database. In general, these include:

- Recoverability - Creating and testing Backups
- Integrity - Verifying or helping to verify data integrity
- Security - Defining and/or implementing access controls to the data
- Availability - Ensuring maximum uptime
- Performance - Ensuring maximum performance
- Development and testing support - Helping programmers and engineers to efficiently utilize the database.

The role of a database administrator has changed according to the technology of database management systems (DBMSs) as well as the needs of the owners of the databases. For example, although logical and physical database designs are traditionally the duties of a database **analyst or database designer, a DBA may be tasked to perform those** duties.

2.0 OBJECTIVES

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

- answer the question of who is a database administrator
- identify the various functions of database administrator
- know the different types of database administration
- understand the nature of database administration
- know the tools used in database administration.

3.0 MAIN CONTENT

3.1 Duties of Database Administrator

The duties of a database administrator vary and depend on the job description, corporate and Information Technology (IT) policies and the technical features and capabilities of the DBMS being administered. They nearly always include disaster recovery (backups and testing of backups), performance analysis and tuning, data dictionary maintenance, and some database design.

Some of the roles of the DBA may include:

•Installation of new software — It is primarily the job of the DBA to install new versions of DBMS software, application software, and other software related to DBMS administration. It is important that the DBA or other IS staff members test this new software before it is moved into a production environment.

•Configuration of hardware and software with the system administrator — In many cases the system software can only be accessed by the system administrator. In this case, the DBA must work closely with the system administrator to perform software installations, and to configure hardware and software so that it functions optimally with the DBMS.

•Security administration — One of the main duties of the DBA is to monitor and administer DBMS security. This involves adding and removing users, administering quotas, auditing, and checking for security problems.

•Data analysis — The DBA will frequently be called on to analyze the data stored in the database and to make recommendations relating to performance and efficiency of that data storage. This might relate to the more effective use of indexes, enabling "Parallel Query" execution, or other DBMS specific features.

•Database design (preliminary) — The DBA is often involved at the preliminary database-design stages. Through the involvement of the DBA, many problems that might occur can be eliminated. The DBA knows the DBMS and system, can point out potential problems, and can help the development team with special performance considerations.

•Data modeling and optimization — by modeling the data, it is possible to optimize the system layout to take the most advantage of the I/O subsystem.

•Responsible for the administration of existing enterprise databases and the analysis, design, and creation of new databases.

- Data modeling, database optimization, understanding and implementation of schemas, and the ability to interpret and write complex SQL queries
- Proactively monitor systems for optimum performance and capacity constraints
- Establish standards and best practices for SQL
- Interact with and coach developers in SQL scripting

Recoverability

Recoverability means that, if a data entry error, program bug or hardware failure occurs, the DBA can bring the database backward in time to its state at an instant of logical consistency before the damage was done. Recoverability activities include making database backups and storing them in ways that minimize the risk that they will be damaged or lost, such as placing multiple copies on removable media and storing them outside the affected area of an anticipated disaster. Recoverability is the DBA's most important concern.

The backup of the database consists of data with timestamps combined with database logs to change the data to be consistent to a particular moment in time. It is possible to make a backup of the database containing only data without timestamps or logs, but the DBA must take the database offline to do such a backup.

The recovery tests of the database consist of restoring the data, then applying logs against that data to bring the database backup to consistency at a particular point in time up to the last transaction in the logs. Alternatively, an offline database backup can be restored simply by placing the data in-place on another copy of the database.

If a DBA (or any administrator) attempts to implement a recoverability plan without the recovery tests, there is no guarantee that the backups are at all valid. In practice, in all but the most mature RDBMS packages, backups rarely are valid without extensive testing to be sure that no bugs or human error have corrupted the backups.

Security

Security means that users' ability to access and change data conforms to the policies of the business and the delegation decisions of its managers. Like other metadata, a relational DBMS manages security information in the form of tables. These tables are the "keys to the kingdom" and so it is important to protect them from intruders. so that is why the security is more and more important for the databases.

Performance

Performance means that the database does not cause unreasonable online response times, and it does not cause unattended programs to run for an unworkable period of time. In complex client/server and three-tier systems, the database is just one of many elements that determine the performance that online users and unattended programs experience. Performance is a major motivation for the DBA to become a generalist and coordinate with specialists in other parts of the system outside of traditional bureaucratic reporting lines.

Techniques for database performance tuning have changed as DBA's have become more sophisticated in their understanding of what causes performance problems and their ability to diagnose the problem.

In the 1990s, DBAs often focused on the database as a whole, and looked at database-wide statistics for clues that might help them find out why the system was slow. Also, the actions DBAs took in their attempts to solve performance problems were often at the global, database level, such as changing the amount of computer memory available to the database, or changing the amount of memory available to any database program that needed to sort data.

DBA's now understand that performance problems initially must be diagnosed, and this is best done by examining individual SQL statements, table process, and system architecture, not the database as a whole. Various tools, some included with the database and some available from third parties, provide a behind the scenes look at how the database is handling the SQL statements, shedding light on what's taking so long.

Having identified the problem, the individual SQL statement can be

Development/Testing Support

Development and testing support is typically what the database administrator regards as his or her least important duty, while results-oriented managers consider it the DBA's most important duty. Support activities include collecting sample production data for testing new and

changed programs and loading it into test databases; consulting with programmers about performance tuning; and making table design changes to provide new kinds of storage for new program functions.
Here are some IT roles that are related to the role of database administrator:

•Application programmer or software engineer
•System administrator
•Data administrator
•Data architect

3.2 Typical Work Activities

The work of database administrator (DBA) varies according to the nature of the employing organization and level of responsibility associated with post. The work may be pure maintenance or it may also involve specializing in database development.

Typical responsibility includes some or all of the following:

- establishing the needs of the users and monitoring users access and security
- monitoring performance and managing parameters to provide fast query responses to ‘front end’ users
- mapping out the conceptual design for a planned database in outline
- considering both back end organization of data and front end accessibility for the end user
- refining the logical design so that it can translated into specific data model
- further refining the physical design to meet systems storage requirements
- installing and testing new versions of the database management system
- maintaining data standards including adherence to the Data Protection Act
- writing database documentation, including data standards, procedures and definitions for the data dictionary (metadata)
- controlling access permissions and privileges
- developing, managing and testing backup recovery plans
- ensuring that storage , archiving, and backup procedures are functioning properly
- capacity planning
- working closely with IT project manager, database programmers, and web developers

• communicating regularly with technical applications and operational staff to ensure database integrity and security
• commissioning and installing new applications

Because of the increasing level of hacking and the sensitive nature of data stored, security and recoverability or disaster recovery has become increasingly important aspects of the work.

3.3 Database Administrations and Automation

Database Administration is the function of managing and maintaining database management systems (DBMS) software. Mainstream DBMS software such as Oracle, IBM DB2 and Microsoft SQL Server need ongoing management. As such, corporations that use DBMS software often hire specialized IT (Information Technology) personnel called Database Administrators or DBAs.

3.3.1 Types of Database Administration

There are three types of DBAs:

1. Systems DBAs (sometimes also referred to as Physical DBAs, Operations DBAs or Production Support DBAs)
2. Development DBAs
3. Application DBAs

Depending on the DBA type, their functions usually vary. Below is a brief description of what different types of DBAs do:

• Systems DBAs usually focus on the physical aspects of database administration such as DBMS installation, configuration, patching, upgrades, backups, restores, refreshes, performance optimization, maintenance and disaster recovery.
• Development DBAs usually focus on the logical and development aspects of database administration such as data model design and maintenance, DDL (data definition language) generation, SQL writing and tuning, coding stored procedures, collaborating with developers to help choose the most appropriate DBMS feature/functionality and other pre-production activities.
• Application DBAs are usually found in organizations that have purchased 3rd party application software such as ERP (enterprise resource planning) and CRM (customer relationship management) systems. Examples of such application software include Oracle Applications, Siebel and PeopleSoft (both now part of Oracle Corp.) and SAP. Application DBAs straddle the fence between the DBMS and the application software and are

responsible for ensuring that the application is fully optimized for the database and vice versa. They usually manage all the application components that interact with the database and carry out activities such as application installation and patching, application upgrades, database cloning, building and running data cleanup routines, data load process management, etc.

While individuals usually specialize in one type of database administration, in smaller organizations, it is not uncommon to find a single individual or group performing more than one type of database administration.

3.3.2 Nature of Database Administration

The degree to which the administration of a database is automated dictates the skills and personnel required to manage databases. On one end of the spectrum, a system with minimal automation will require significant experienced resources to manage; perhaps 5-10 databases per DBA. Alternatively an organization might choose to automate a significant amount of the work that could be done manually therefore reducing the skills required to perform tasks. As automation increases, the personnel needs of the organization splits into highly skilled workers to create and manage the automation and a group of lower skilled "line" DBAs who simply execute the automation.

Database administration work is complex, repetitive, time-consuming and requires significant training. Since databases hold valuable and mission-critical data, companies usually look for candidates with multiple years of experience. Database administration often requires DBAs to put in work during off-hours (for example, for planned after hours downtime, in the event of a database-related outage or if performance has been severely degraded). DBAs are commonly well compensated for the long hours.

3.3.3 Database Administration Tools

Often, the DBMS software comes with certain tools to help DBAs manage the DBMS. Such tools are called native tools. For example, Microsoft SQL Server comes with SQL Server Enterprise Manager and Oracle has tools such as SQL*Plus and Oracle Enterprise Manager/Grid Control. In addition, 3rd parties such as BMC, Quest Software, Embarcadero and SQL Maestro Group offer GUI tools to monitor the DBMS and help DBAs carry out certain functions inside the database more easily.

Another kind of database software exists to manage the provisioning of new databases and the management of existing databases and their related resources. The process of creating a new database can consist of hundreds or thousands of unique steps from satisfying prerequisites to configuring backups where each step must be successful before the next can start. A human cannot be expected to complete this procedure in the same exact way time after time - exactly the goal when multiple databases exist. As the number of DBAs grows, without automation the number of unique configurations frequently grows to be costly/difficult to support. All of these complicated procedures can be modeled by the best DBAs into database automation software and executed by the standard DBAs. Software has been created specifically to improve the reliability and repeatability of these procedures such as Stratavia's Data Palette and GridApp Systems Clarity.

3.3.4 The Impact of IT Automation on Database Administration

Recently, automation has begun to impact this area significantly. Newer technologies such as HP/Opsware's SAS (Server Automation System) and Stratavia's Data Palette suite have begun to increase the automation of servers and databases respectively causing the reduction of database related tasks. However at best this only reduces the amount of mundane, repetitive activities and does not eliminate the need for DBAs. The intention of DBA automation is to enable DBAs to focus on more proactive activities around database architecture and deployment.

3.3.5 Learning Database Administration

There are several education institutes that offer professional courses, including late-night programs, to allow candidates to learn database administration. Also, DBMS vendors such as Oracle, Microsoft and IBM offer certification programs to help companies to hire qualified DBA practitioners.

4.0 CONCLUSION

Database management system (DBMS) is so important in an organization that a special manager is often appointed to oversee its activities. The database administrator is responsible for the installation and coordination of DBMS. They are responsible for managing one of the most valuable resources of any organization, its data. The database administrator must have a sound knowledge of the structure of the database and of the DBMS. The DBA must be thoroughly conversant with the organization, it's system and the information need of managers.

5.0 SUMMARY

- A Database administrator (DBA) is a person who is responsible for the environmental aspects of a database
- The duties of a database administrator vary and depend on the job description, corporate and Information Technology (IT) policies and the technical features and capabilities of the DBMS being administered. They nearly always include disaster recovery (backups and testing of backups), performance analysis and tuning, data dictionary maintenance, and some database design.
- Techniques for database performance tuning have changed as DBA's have become more sophisticated in their understanding of what causes performance problems and their ability to diagnose the problem
- The work of database administrator (DBA) varies according to the nature of the employing organization and level of responsibility associated with post.
- Database Administration is the function of managing and maintaining DATABASE MANAGEMENT APPLICATION SYSTEMs (DBMS) software.
- The degree to which the administration of a database is automated dictates the skills and personnel required to manage databases

6.0 TUTOR-MARKED ASSIGNMENT

1. Mention 5 roles of database administrator
2. Mention the types of database administrations

7.0 REFERENCES/FURTHER READINGS

Association for Computing Machinery SIGIR Forum archive Volume 7, Issue 4.

The Origins of the Data Base Concept, Early DBMS Systems including DS and IMS, the Data Base Task Group, and the Hierarchical, Network and Relational Data Models are discussed in Thomas Haigh, "'A Veritable Bucket of Facts:' Origins of the Data Base Management System," ACM SIGMOD Record 35:2 (June 2006).

How Database Systems Share Storage.

MODULE 3

UNIT 1 RELATIONAL DATABASE MANAGEMENT SYSTEMS

CONTENTS

1.0 INTRODUCTION

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd. Most popular commercial and open source databases currently in use are based on the relational model.

A short definition of an RDBMS may be a DBMS in which data is stored in the form of tables and the relationship among the data is also stored in the form of tables.

2.0 OBJECTIVES

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

- define relational DATABASE MANAGEMENT APPLICATION SYSTEM
- trace the origin and development of RDBMS
- identify the market structure of RDBMS
- identify the major types of relational management systems
- compare and contrast the types of RDBMS based on several criteria

3.0 MAIN CONTENT

3.1 History of the Term

E. F. Codd introduced the term in his seminal paper "A Relational Model of Data for Large Shared Data Banks", published in 1970. In this paper and later papers he defined what he meant by relational. One well-known definition of what constitutes a relational database system is Codd's 12 rules. However, many of the early implementations of the relational model did not conform to all of Codd's rules, so the term gradually came to describe a broader class of database systems. At a minimum, these systems:

- presented the data to the user as relations (a presentation in tabular form, i.e. as a collection of tables with each table consisting of a set of rows and columns, can satisfy this property)
- provided relational operators to manipulate the data in tabular form

The first systems that were relatively faithful implementations of the relational model were from the University of Michigan; Micro DBMS (1969) and from IBM UK Scientific Centre at Peterlee; IS1 (1970–72) and its followon PRTV (1973–79). The first system sold as an RDBMS was Multics Relational Data Store, first sold in 1978. Others have been Berkeley Ingres QUEL and IBM BS12.

The most popular definition of an RDBMS is a product that presents a view of data as a collection of rows and columns, even if it is not based strictly upon relational theory. By this definition, RDBMS products typically implement some but not all of Codd's 12 rules.

A second, theory-based school of thought argues that if a database does not implement all of Codd's rules (or the current understanding on the relational model, as expressed by Christopher J Date, Hugh Darwen and others), it is not relational. This view, shared by many theorists and other strict adherents to Codd's principles, would disqualify most DBMSs as not relational. For clarification, they often refer to some RDBMSs as Truly-Relational Database Management Systems (TRDBMS), naming others Pseudo-Relational Database Management *Systems (PRDBMS).*

Almost all commercial relational DBMSs employ SQL as their query language. Alternative query languages have been proposed and implemented, but very few have become commercial products.

3.2 Market Structure

Given below is a list of top RDBMS vendors in 2006 with figures in millions of United States Dollars published in an IDC study.

Vendor	Global Revenue
Oracle	7,912
IBM	3,483
Microsoft	3,052
Sybase	524
Teradata	457
Others	1,624
Total	**16,452**

Low adoption costs associated with open-source RDBMS products such as MySQL and PostgreSQL have begun influencing vendor pricing and licensing strategies].

3.3 Features and Responsibilities of an RDBMS

As mentioned earlier, an RDBMS is software that is used for creating and maintaining a database. Maintaining involves several tasks that an RDBMS takes care of. These tasks are as follow:

Control Data Redundancy

Since data in an RDBMS is spread across several tables, repetition or redundancy is reduced. Redundant data can be extracted and stored in another table, along with a field that is common to both the tables. Data can then be extracted from the two tables by using the common field.

Data Abstraction

This would imply that the RDBMS hides the actual way, in which data is stored, while providing the user with a conceptual representation of the data.

Support for Multiple Users

A true RDBMS allows effective sharing of data. That is, it ensures that several users can concurrently access the data in the database without affecting the speed of the data access.

In a database application, which can be used by several users concurrently, there is the possibility that two users may try to modify a particular record at the same time. This could lead to one person's changes being made while the others are overwritten. To avoid such confusion, most RDBMSs provide a record-locking mechanism. This mechanism ensures that no two users could modify a particular record at the same time. A record is as it were "locked" while one user makes changes to it. Another user is therefore not allowed to modify it till the changes are complete and the record is saved. The "lock" is then released, and the record available for editing again.

Multiple Ways of Interfering to the System

This would require the database to be able to be accessible through different query languages as well as programming languages. It would also mean that a variety of front-end tools should be able to use the database as a back-end. For example data stored in Microsoft Access can be displayed and manipulated using forms created in software such as Visual Basic or Front Page 2000.

Restricting Unauthorized Access

An RDBMS provides a security mechanism that ensures that data in the database is protected from unauthorized access and malicious use. The security that is implemented in most RDBMSs is referred to as 'User-level security', wherein the various users of the database are assigned usernames and passwords., only when the user enters the correct username and password is he able to access the data in the database.

In addition to this, a particular user could be restricted to only view the data, while another could have the rights to modify the data. A third user could have right s to change the structure of some table itself, in addition to the rights that the other two have.
When security is implemented properly, data is secure and cannot be tampered with.

Enforcing Integrity Constraints

RDBMS provide a set of rules that ensure that data entered into a table is valid. These rules must remain true for a database to preserve integrity. 'Integrity constraints' are specified at the time of creating the database, and are enforced by the RDBMS.

For example in a 'Marks 'table, a constraint can be added to ensure that the marks in each subject be between 0 and 100. Such a constraint is called a 'Check' constraint. It is a rule that can be set by the user to ensure that only data that meets the criteria specified there is allowed to enter the database. The given example ensures that only a between 0 and 100 can be entered into the marks column.

Backup and Recovery

In spite of ensuring that the database is secure from unauthorized access/ user as well as invalid entries, there is always a danger that the data in the database could get lost. They could happen due to some hardware problems or system crash. It could therefore result in a loss of all data. To guard the database from this, most RDBMSs have inbuilt backup and recovery techniques that ensure that the database is protected from these kinds of fatalities too.

3.4 Comparison of Relational Database Management Systems

The following tables compare general and technical information for a number of relational database management systems. Comparisons are based on the stable versions without any add-ons, extensions or external programs.

3.4.1 General information

	Maintainer	First public release date	Latest stable version	Software license
4th Dimension	4D s.a.s	1984	v11 SQL	Proprietary
ADABAS	Software AG	1970	?	?
Adaptive Server Enterprise	Sybase	1987	15.0	Proprietary
Advantage Database Server	Sybase	1992	8.1	Proprietary
Apache Derby	Apache	2004	10.4.1.3	Apache License
Datacom	CA	?	11.2	Proprietary
DB2	IBM	1982	9.5	Proprietary
DBISAM	Elevate Software	?	4.25	Proprietary
Datawasp	Significant Data Systems	April 2008	1.0.1	Proprietary
ElevateDB	Elevate Software	?	1.01	Proprietary
FileMaker	FileMaker	1984	9	proprietary
Firebird	Firebird project	July 25, 2000	2.1.0	IPL and IDPL
Informix	IBM	1985	11.10	Proprietary

HSQLDB	HSQL Development Group	2001	1.8.0	BSD
H2	H2 Software	2005	1.0	EPL and modified MPL
Ingres	Ingres Corp.	1974	Ingres 2006 r2 9.1.0	GPL and proprietary
InterBase	CodeGear	1985	2007	Proprietary
MaxDB	SAP AG	?	7.6	GPL or proprietary
Microsoft Access	Microsoft	1992	12 (2007)	Proprietary
Microsoft Visual Foxpro	Microsoft	?	9 (2005)	Proprietary
Microsoft SQL Server	Microsoft	1989	9.00.3042 (2005 SP2)	Proprietary
MonetDB	The MonetDB Developer Team	2004	4.16 (Feb. 2007)	MonetDB Public License v1.1
MySQL	Sun Microsystems	November 1996	5.0.67	GPL or proprietary
HP NonStop SQL	Hewlett-Packard	1987	SQL MX 2.0	Proprietary
Omnis Studio	TigerLogic Inc	July 1982	4.3.1 Release 1 (May 2008)	Proprietary
Oracle	Oracle Corporation	November 1979	11g Release 1 (September 2007)	Proprietary
Oracle Rdb	Oracle Corporation	1984	7.2	Proprietary
OpenEdge	Progress Software Corporation	1984	10.1C	Proprietary
OpenLink Virtuoso	OpenLink Software	1998	5.0.5 (January 2008)	GPL or proprietary
Pervasive PSQL	Pervasive Software	?	9	Proprietary
Polyhedra DBMS	ENEA AB	1993	8.0 (July 2008)	Proprietary
PostgreSQL	PostgreSQL Global Development Group	June 1989	8.3.3 (12 June 2008)	BSD
Pyrrho DBMS	University of Paisley	November 2005	0.5	Proprietary
RBase	RBase	?	7.6	Proprietary

RDM Embedded	**Birdstep** Technology	1984	8.1	Proprietary
RDM Server	**Birdstep** Technology	1990	8.0	Proprietary
ScimoreDB	**Scimore**	**2005**	**2.5**	**Freeware**
SmallSQL	**SmallSQL**	**April 16,** 2005	0.19	LGPL
SQL Anywhere	**Sybase**	**1992**	**10.0**	**Proprietary**
SQLite	**D. Richard** Hipp	August 17, 2000	3.5.7 (17 March 2008)	Public domain
Teradata	**Teradata**	**1984**	**V12**	**Proprietary**
Valentina	**Paradigma** Software	February 1998	3.0.1	Proprietary

3.4.2 Operating system support

The operating systems the RDBMSes can run on.

	Windows	**Mac OS X**	**Linux**	**BSD**	**UNIX**	**z/OS 1**
4th Dimension	**Yes**	**Yes**	**No**	**No**	**No**	**No**
ADABAS	**Yes**	**No**	**Yes**	**No**	**Yes**	**Yes**
Adaptive Server Enterprise	**Yes**	**No**	**Yes**	**Yes**	**Yes**	**No**
Advantage Database Server	**Yes**	**No**	**Yes**	**No**	**No**	**No**
Apache Derby 2	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
DataCom	**No**	**No**	**No**	**No**	**No**	**Yes**
Datawasp	**Yes**	**No**	**No**	**No**	**No**	**No**
DB2 5	**Yes**	**No**	**Yes**	**No**	**Yes**	**Yes**
Firebird	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Maybe**
HSQLDB 2	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
H2 2	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Maybe**
FileMaker	**Yes**	**Yes**	**No**	**No**	**No**	**No**
Informix	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**No**
Ingres	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Partial**
InterBase	**Yes**	**Yes**	**Yes**	**No**	**Yes** (Solaris)	No
MaxDB	**Yes**	**No**	**Yes**	**No**	**Yes**	**Maybe**
Microsoft Access	**Yes**	**No**	**No**	**No**	**No**	**No**
Microsoft Visual Foxpro	**Yes**	**No**	**No**	**No**	**No**	**No**
Microsoft SQL Server	**Yes**	**No**	**No**	**No**	**No**	**No**
MonetDB	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**No**
MySQL	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Maybe**
Omnis Studio	**Yes**	**Yes**	**Yes**	**No**	**No**	**No**

Oracle	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**
Oracle Rdb 3	**No**	**No**	**No**	**No**	**No**	**No**
OpenEdge	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
OpenLink Virtuoso	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Polyhedra DBMS	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
PostgreSQL	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**No**
Pyrrho DBMS	**Yes (.NET)**	**No**	**Yes** (Mono)	No	No	No
RBase	**Yes**	**No**	**No**	**No**	**No**	**No**
RDM Embedded	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**No**
RDM Server	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**No**
ScimoreDB	**Yes**	**No**	**No**	**No**	**No**	**No**
SmallSQL 2	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
SQL Anywhere	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**No**
SQLite	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Maybe**
Teradata	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
Valentina	**Yes**	**Yes**	**Yes**	**No**	**No**	**No**

Note (1): Open source databases listed as UNIX-compatible will likely compile and run under z/OS's built-in UNIX System Services (USS) subsystem. Most databases listed as Linux-compatible can run alongside z/OS on the same server using Linux on zSeries.

Note (2: The database availability depends on Java Virtual Machine not on the operatin system

Note (3): Oracle Rdb was originally developed by DEC, and runs on OpenVMS

Note (4): Oracle database 11g also runs on OpenVMS, HP/UX and AIX. 10g also supported BS2000/OSD and z/OS (31-bit), but that support has been discontinued in 11g. Earlier versions than 10g were available on a wide variety of platforms.

Note (5): DB2 is also available for i5/OS, z/VM, z/VSE. Previous versions were also available for OS/2.

3.4.3 Fundamental features

Information about what fundamental RDBMS features are implemented natively.

	ACID	**Referential integrity**	**Transactions**	**Unicode**	**Interface**
4th Dimension	**Yes**	**Yes**	**Yes**	**Yes**	**GUI & SQL**
ADABAS	**?**	**?**	**?**	**?**	**?**
Adaptive Server Enterprise	Yes	Yes	Yes	Yes	?

Advantage Database Server	Yes	Yes	Yes	No	API & SQL
Apache Derby	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
Datawasp	**No**	**Yes**	**Yes**	**Yes**	**GUI**
DB2	**Yes**	**Yes**	**Yes**	**Yes**	**GUI & SQL**
Firebird	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
HSQLDB	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
H2	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
Informix	**Yes**	**Yes**	**Yes**	**Yes**	**?**
Ingres	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
InterBase	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
MaxDB	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
Microsoft Access	**No**	**Yes**	**Yes**	**Yes**	**GUI & SQL**
Microsoft Visual Foxpro	**No**	**Yes**	**Yes**	**No**	**GUI & SQL**
Microsoft SQL Server	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
MonetDB	**Yes**	**Yes**	**Yes**	**Yes**	**?**
MySQL	**Yes 6**	**Yes 6**	**Yes 6**	**Partial**	**SQL**
Oracle	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
Oracle Rdb	**Yes**	**Yes**	**Yes**	**Yes**	**?**
OpenEdge	**Yes**	**No 7**	**Yes**	**Yes**	**Progress 4GL** & SQL
OpenLink Virtuoso	**Yes**	**Yes**	**Yes**	**Yes**	**?**
Polyhedra DBMS	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
PostgreSQL	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
Pyrrho DBMS	**Yes**	**Yes**	**Yes**	**Yes**	**?**
RDM Embedded	**Yes**	**Yes**	**Yes**	**Yes**	**SQL & API**
RDM Server	**Yes**	**Yes**	**Yes**	**Yes**	**SQL & API**
ScimoreDB	**Yes**	**Yes**	**Yes**	**Partial**	**SQL**
SQL Anywhere	**Yes**	**Yes**	**Yes**	**Yes**	**?**
SQLite	**Yes**	**No 8**	**Basic 8**	**Yes**	**SQL**
Teradata	**Yes**	**Yes**	**Yes**	**Yes**	**SQL**
Valentina	**No**	**Yes**	**No**	**Yes**	**?**

Note (6): For transactions and referential integrity, the InnoDB table type must be used; Windows installer sets this as default if support for transactions is selected, on other operating systems the default table type is MyISAM. However, even the InnoDB table type permits storage of values that exceed the data range; some view this as violating the integrity constraint of ACID.

Note (7): FOREIGN KEY constraints are parsed but are not enforced. Triggers can be used instead. Nested transactions are not supported.
Note (8): Available via Triggers.

4.0 CONCLUSION

The most dominant model in use today is the relational database management systems, usually used with the structured query language SQL query language. Many DBMS also support the Open Database Connectivitry that supports a standard way for programmers to access the DATABASE MANAGEMENT APPLICATION SYSTEMs.

5.0 SUMMARY

- A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd. Most popular commercial and open source databases currently in use are based on the relational model.
- E. F. Codd introduced the term in his seminal paper "A Relational Model of Data for Large Shared Data Banks", published in 1970. In this paper and later papers he defined what he meant by relational. One well-known definition of what constitutes a relational database system is Codd's 12 rules
- The most popular definition of an RDBMS is a product that presents a view of data as a collection of rows and columns, even if it is not based strictly upon relational theory
- As mentioned earlier, an RDBMS is software that is used for creating and maintaining a database. Maintaining involves several tasks that an RDBMS takes care of
- Comparisons are based on the stable versions without any add-ons, extensions or external programs.

6.0 TUTOR-MARKED ASSIGNMENT

1. List 5 features of Relational DATABASE MANAGEMENT APPLICATION SYSTEMs
2. Mention 5 criteria you can use to differentiate types of RDBMSs

7.0 REFERENCES/FURTHER READINGS

Comparison of different SQL implementations against SQL standards. Includes Oracle, DB2, Microsoft SQL Server, MySQL and PostgreSQL. (08/Jun/2007).

Comparison of Oracle 8/9i, MySQL 4.x and PostgreSQL 7.x DBMS against SQL standards. (14/Mar/2005).

Comparison of Oracle and SQL Server. (2004).

Comparison of geometrical data handling in PostgreSQL, MySQL and DB2 (29/Sep/2003).

Open Source Database Software Comparison (Mar/2005).

PostgreSQL vs. MySQL vs. Commercial Databases: It's All About What You Need (12/Apr/2004).

UNIT 2 DATA WAREHOUSE

CONTENTS

1.0 INTRODUCTION

A data warehouse is a repository of an organization's electronically stored data. Data warehouses are designed to facilitate reporting and analysis.

This classic definition of the data warehouse focuses on data storage. However, the means to retrieve and analyze data, to extract, transform and load data, and to manage the dictionary data are also considered essential components of a data warehousing system. Many references to data warehousing use this broader context. Thus, an expanded definition for data warehousing includes business intelligence tools, tools to extract, transform, and load data into the repository, and tools to manage and retrieve metadata.

In contrast to data warehouses are operational systems which perform day-to-day transaction processing.

2.0 OBJECTIVES

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

- define data warehouse
- trace the history and development process of data warehouse
- list various benefits of data warehouse
- define the architecture of a data warehouse
- compare and contrast Data Warehouses and Operational Systems
- know what is a data warehouse appliance, and the disadvantages of data warehouse
- have idea of what the future holds for data warehouse concept.

3.0 MAIN CONTENT

3.1 History

The concept of data warehousing dates back to the late-1980s when IBM researchers Barry Devlin and Paul Murphy developed the "business data warehouse". In essence, the data warehousing concept was intended to provide an architectural model for the flow of data from operational systems to decision support environments. The concept attempted to address the various problems associated with this flow - mainly, the high costs associated with it. In the absence of a data warehousing architecture, an enormous amount of redundancy of information was required to support the multiple decision support environment that usually existed. In larger corporations it was typical for multiple decision support environments to operate independently. Each environment served different users but often required much of the same data. The process of gathering, cleaning and integrating data from various sources, usually long existing operational systems (usually referred to as legacy systems), was typically in part replicated for each environment. Moreover, the operational systems were frequently reexamined as new decision support requirements emerged. Often new requirements necessitated gathering, cleaning and integrating new data from the operational systems that were logically related to prior gathered data.

Based on analogies with real-life warehouses, data warehouses were intended as large-scale collection/storage/staging areas for corporate data. Data could be retrieved from one central point or data could be distributed to "retail stores" or "data marts" which were tailored for ready access by users.

3.2 Benefits of Data Warehousing

Some of the benefits that a data warehouse provides are as follows:

- A data warehouse provides a common data model for all data of interest regardless of the data's source. This makes it easier to report and analyze information than it would be if multiple data models were used to retrieve information such as sales invoices, order receipts, general ledger charges, etc.
- Prior to loading data into the data warehouse, inconsistencies are identified and resolved. This greatly simplifies reporting and analysis.
- Information in the data warehouse is under the control of data warehouse users so that, even if the source system data is purged over time, the information in the warehouse can be stored safely for extended periods of time.
- Because they are separate from operational systems, data warehouses provide retrieval of data without slowing down operational systems.
- Data warehouses facilitate decision support system applications such as trend reports (e.g., the items with the most sales in a particular area within the last two years), exception reports, and reports that show actual performance versus goals.
- Data warehouses can work in conjunction with and, hence, enhance the value of operational business applications, notably customer relationship management (CRM) systems.

3.3 Data Warehouse Architecture

Architecture, in the context of an organization's data warehousing efforts, is a conceptualization of how the data warehouse is built. There is no right or wrong architecture. The worthiness of the architecture can be judged in how the conceptualization aids in the building, maintenance, and usage of the data warehouse.

One possible simple conceptualization of a data warehouse architecture consists of the following interconnected layers:

Operational Database Layer

The source data for the data warehouse - An organization's ERP systems fall into this layer.

Informational Access Layer

The data accessed for reporting and analyzing and the tools for reporting and analyzing data - Business intelligence tools fall into this layer. And the Inmon-Kimball differences about design methodology, discussed later in this article, have to do with this layer.

Data access Layer

The interface between the operational and informational access layer - Tools to extract, transform, load data into the warehouse fall into this layer.

Metadata Layer

The data directory - This is often usually more detailed than an operational system data directory. There are dictionaries for the entire warehouse and sometimes dictionaries for the data that can be accessed by a particular reporting and analysis tool.

3.4 Normalized Versus Dimensional Approach to Storage of Data

There are two leading approaches to storing data in a data warehouse - the dimensional approach and the normalized approach.

In the dimensional approach, transaction data are partitioned into either “facts”, which are generally numeric transaction data, or "dimensions", which are the reference information that gives context to the facts. For example, a sales transaction can be broken up into facts such as the number of products ordered and the price paid for the products, and into dimensions such as order date, customer name, product number, order ship-to and bill-to locations, and salesperson responsible for receiving the order. A key advantage of a dimensional approach is that the data warehouse is easier for the user to understand and to use. Also, the retrieval of data from the data warehouse tends to operate very quickly. The main disadvantages of the dimensional approach are: 1) In order to maintain the integrity of facts and dimensions, loading the data warehouse with data from different operational systems is complicated, and 2) It is difficult to modify the data warehouse structure if the organization adopting the dimensional approach changes the way in which it does business.

In the normalized approach, the data in the data warehouse are stored following, to a degree, the Codd normalization rule. Tables are grouped together by subject areas that reflect general data categories (e.g., data on customers, products, finance, etc.) The main advantage of this approach is that it is straightforward to add information into the database. A disadvantage of this approach is that, because of the number of tables involved, it can be difficult for users both to 1) join data from different sources into meaningful information and then 2) access the information without a precise understanding of the sources of data and of the data structure of the data warehouse.

These approaches are not exact opposites of each other. Dimensional approaches can involve normalizing data to a degree.

3.5 Conforming Information

Another important decision in designing a data warehouse is which data to conform and how to conform the data. For example, one operational system feeding data into the data warehouse may use "M" and "F" to denote sex of an employee while another operational system may use "Male" and "Female". Though this is a simple example, much of the work in implementing a data warehouse is devoted to making similar meaning data consistent when they are stored in the data warehouse. Typically, extract, transform, load tools are used in this work.

3.6 Top-Down versus Bottom-Up Design Methodologies

Bottom-Up Design

Ralph Kimball, a well-known author on data warehousing, is a proponent of the bottom-up approach to data warehouse design. In the bottom-up approach data marts are first created to provide reporting and analytical capabilities for specific business processes. Data marts contain atomic data and, if necessary, summarized data. These data marts can eventually be unioned together to create a comprehensive data warehouse. The combination of data marts is managed through the implementation of what Kimball calls "a data warehouse bus architecture".

Business value can be returned as quickly as the first data marts can be created. Maintaining tight management over the data warehouse bus architecture is fundamental to maintaining the integrity of the data warehouse. The most important management task is making sure dimensions among data marts are consistent. In Kimball words, this means that the dimensions "conform".

Top-Down Design

Bill Inmon, one of the first authors on the subject of data warehousing, has defined a data warehouse as a centralized repository for the entire enterprise. Inmon is one of the leading proponents of the top-down approach to data warehouse design, in which the data warehouse is designed using a normalized enterprise data model. "Atomic" data, that is, data at the lowest level of detail, are stored in the data warehouse. Dimensional data marts containing data needed for specific business processes or specific departments are created from the data warehouse. In the Inmon vision the data warehouse is at the center of the "Corporate Information Factory" (CIF), which provides a logical framework for delivering business intelligence (BI) and business management capabilities. The CIF is driven by data provided from business operations

Inmon states that the data warehouse is:

Subject-Oriented

The data in the data warehouse is organized so that all the data elements relating to the same real-world event or object are linked together.

Time-Variant

The changes to the data in the data warehouse are tracked and recorded so that reports can be produced showing changes over time.

Non-Volatile

Data in the data warehouse is never over-written or deleted - once committed, the data is static, read-only, and retained for future reporting.

Integrated

The data warehouse contains data from most or all of an organization's operational systems and this data is made consistent.

The top-down design methodology generates highly consistent dimensional views of data across data marts since all data marts are loaded from the centralized repository. Top-down design has also proven to be robust against business changes. Generating new dimensional data marts against the data stored in the data warehouse is a relatively simple task. The main disadvantage to the top-down methodology is that it represents a very large project with a very broad scope. The up-front cost for implementing a data warehouse using the top-down methodology is significant, and the duration of time from the start of project to the point that end users experience initial benefits can be substantial. In addition, the top-down methodology can be inflexible and unresponsive to changing departmental needs during the implementation phases.

Hybrid Design

Over time it has become apparent to proponents of bottom-up and top-down data warehouse design that both methodologies have benefits and risks. Hybrid methodologies have evolved to take advantage of the fast turn-around time of bottom-up design and the enterprise-wide data consistency of top-down design

3.7 Data Warehouses versus Operational Systems

Operational systems are optimized for preservation of data integrity and speed of recording of business transactions through use of database normalization and an entity-relationship model. Operational system designers generally follow the Codd rules of data normalization in order to ensure data integrity. Codd defined five increasingly stringent rules of normalization. Fully normalized database designs (that is, those satisfying all five Codd rules) often result in information from a business transaction being stored in dozens to hundreds of tables. Relational databases are efficient at managing the relationships between these tables. The databases have very fast insert/update performance because only a small amount of data in those tables is affected each time a transaction is processed. Finally, in order to improve performance, older data are usually periodically purged from operational systems.

Data warehouses are optimized for speed of data retrieval. Frequently data in data warehouses are denormalised via a dimension-based model. Also, to speed data retrieval, data warehouse data are often stored multiple times - in their most granular form and in summarized forms called aggregates. Data warehouse data are gathered from the operational systems and held in the data warehouse even after the data has been purged from the operational systems.

3.8 Evolution in Organization Use of Data Warehouses

Organizations generally start off with relatively simple use of data warehousing. Over time, more sophisticated use of data warehousing evolves. The following general stages of use of the data warehouse can be distinguished:

Off line Operational Databases

Data warehouses in this initial stage are developed by simply copying the data of an operational system to another server where the processing load of reporting against the copied data does not impact the operational system's performance.

Off line Data Warehouse

Data warehouses at this stage are updated from data in the operational systems on a regular basis and the data warehouse data is stored in a data structure designed to facilitate reporting.

Real Time Data Warehouse

Data warehouses at this stage are updated every time an operational system performs a transaction (e.g., an order or a delivery or a booking.)

Integrated Data Warehouse

Data warehouses at this stage are updated every time an operational system performs a transaction. The data warehouses then generate transactions that are passed back into the operational systems.

3.9 Disadvantages of Data Warehouses

There are also disadvantages to using a data warehouse. Some of them are:

- Over their life, data warehouses can have high costs. The data warehouse is usually not static. Maintenance costs are high.
- Data warehouses can get outdated relatively quickly. There is a cost of delivering suboptimal information to the organization.
- There is often a fine line between data warehouses and operational systems. Duplicate, expensive functionality may be developed. Or, functionality may be developed in the data warehouse that, in retrospect, should have been developed in the operational systems and vice versa.

3.10 Data Warehouse Appliance

A data warehouse appliance is an integrated set of servers, storage, OS, DBMS and software specifically pre-installed and pre-optimized for data warehousing. Alternatively, the term is also used for similar software-only systems that purportedly are very easy to install on specific recommended hardware configurations. DW appliances provide solutions for the mid-to-large volume data warehouse market, offering low-cost performance most commonly on data volumes in the terabyte to petabyte range.

Technology Primer

Most DW appliance vendors use massively parallel processing (MPP) architectures to provide high query performance and platform scalability. MPP architectures consist of independent processors or servers executing in parallel. Most MPP architectures implement a "shared nothing architecture" where each server is self-sufficient and controls its own memory and disk. Shared nothing architectures have a proven record for high scalability and little contention. DW appliances distribute data onto dedicated disk storage units connected to each server in the appliance. This distribution allows DW appliances to resolve a relational query by scanning data on each server in parallel. The divide-and-conquer approach delivers high performance and scales linearly as new servers are added into the architecture.

MPP database architectures are not new. Teradata, Tandem, Britton Lee, and Sequent offered MPP SQL-based architectures in the 1980s. The re-emergence of MPP data warehouses has been aided by open source and commodity components. Advances in technology have reduced costs and improved performance in storage devices, multi-core CPUs and networking components. Open source RDBMS products, such as Ingres and PostgreSQL, reduce software license costs and allow DW appliance vendors to focus on optimization rather than providing basic database functionality. Open source Linux provides a stable, well-implemented OS for DW appliances.

History

Many consider Teradata's initial product as the first DW appliance (or Britton-Lee's, but Britton Lee—renamed ShareBase—was acquired by Teradata in June, 1990). Some regard Teradata's current offerings as still being other appliances, while others argue that they fall short in ease of installation or administration. Interest in the data warehouse appliance category is generally dated to the emergence of Netezza in the early 2000s.

More recently, a second generation of modern DW appliances has emerged, marking the move to mainstream vendor integration. IBM integrated its InfoSphere Warehouse (formerly DB2 Warehouse) with its own servers and storage to create the IBM InfoSphere Balanced Warehouse. Other DW appliance vendors have partnered with major hardware vendors to help bring their appliances to market. DATAllegro partners with EMC and Dell and implements open source Ingres on Linux. Greenplum has a partnership with Sun Microsystems and implements Bizgres (a form of PostgreSQL) on Solaris using the ZFS file system. HP Neoview has a wholly-owned solution and uses HP NonStop SQL.

Kognitio offers a row-based "virtual" data warehouse appliance while Vertica, and ParAccel offer column-based "virtual" data warehouse appliances. Like Greenplum, ParAccel partners with Sun Microsystems. These solutions provide software-only solutions deployed on clusters of commodity hardware. Kognitio's homegrown WX2 database runs on several blade configurations. Other players in the DW appliance space include Calpont and Dataupia.

Recently, the market has seen the emergence of data warehouse bundles where vendors combine their hardware and database software together as a data warehouse platform. The Oracle Optimized Warehouse Initiative combines the Oracle Database with the industry's leading computer manufacturers Dell, EMC, HP, IBM, SGI and Sun Microsystems. Oracle's Optimized Warehouses are pre-validated configurations and the database software comes pre-installed, though some analysts differ as to whether these should be regarded as appliances.

Benefits

Reduction in Costs

The total cost of ownership (TCO) of a data warehouse consists of initial entry costs, on-going maintenance costs and the cost of increasing capacity as the data warehouse grows. DW appliances offer low entry and maintenance costs. Initial costs range from $10,000 to $150,000 per terabyte, depending on the size of the DW appliance installed.

The resource cost for monitoring and tuning the data warehouse makes up a large part of the TCO, often as much as 80%. DW appliances reduce administration for day-to-day operations, setup and integration. Many also offer low costs for expanding processing power and capacity.

With the increased focus on controlling costs combined with tight IT Budgets, data warehouse managers need to reduce and manage expenses while leveraging their technology as much as possible making DW appliances a natural solution.

Parallel Performance

DW appliances provide a compelling price/performance ratio. Many support mixed-workloads where a broad range of ad-hoc queries and reports run simultaneously with loading. DW appliance vendors use several distribution and partitioning methods to provide parallel performance. Some DW appliances scan data using partitioning and sequential I/O instead of index usage. Other DW appliances use

standard database indexing.
With high performance on highly granular data, DW appliances are able to address analytics that previously could not meet performance requirements.

Reduced Administration

DW appliances provide a single vendor solution and take ownership for optimizing the parts and software within the appliance. This eliminates the customer's costs for integration and regression testing of the DBMS, storage and OS on a terabyte scale and avoids some of the compatibility issues that arise from multi-vendor solutions. A single support point also provides a single source for problem resolution and a simplified upgrade path for software and hardware.

The care and feeding of DW appliances is less than many alternate data warehouse solutions. DW appliances reduce administration through automated space allocation, reduced index maintenance and in most cases, reduced tuning and performance analysis.

Built-in High Availability

DW appliance vendors provide built-in high availability through redundancy on components within the appliance. Many offer warm-standby servers, dual networks, dual power supplies, disk mirroring with robust failover and solutions for server failure.

Scalability

DW appliances scale for both capacity and performance. Many DW appliances implement a modular design that database administrators can add to incrementally, eliminating up-front costs for over-provisioning. In contrast, architectures that do not support incremental expansion result in hours of production downtime, during which database administrators export and re-load terabytes of data. In MPP architectures, adding servers increases performance as well as capacity. This is not always the case with alternate solutions.

Rapid Time-to-Value

Companies increasingly expect to use business analytics to improve the current cycle. DW appliances provide fast implementations without the need for regression and integration testing. Rapid prototyping is possible because of reduced tuning and index creation, fast loading and reduced needs for aggregation in some cases.

Application Uses

DW appliances provide solutions for many analytic application uses, including:

•Enterprise data warehousing
•Super-sized sandboxes isolate power users with resource intensive queries
•Pilot projects or projects requiring rapid prototyping and rapid time-to-value
•Off-loading projects from the enterprise data warehouse; ie large analytical query projects that affect the overall workload of the enterprise data warehouse
•Applications with specific performance or loading requirements
•Data marts that have outgrown their present environment
•Turnkey data warehouses or data marts
•Solutions for applications with high data growth and high performance requirements
•Applications requiring data warehouse encryption

Trends

The DW appliance market is shifting trends in many areas as it evolves:

•Vendors are moving toward using commodity technologies rather than proprietary assembly of commodity components.
•Implemented applications show usage expansion from tactical and data mart solutions to strategic and enterprise data warehouse use.
•Mainstream vendor participation is now apparent.
•With a lower total cost of ownership, reduced maintenance and high performance to address business analytics on growing data volumes, most analysts believe that DW appliances will gain market share.

3.11 The Future of Data Warehousing

Data warehousing, like any technology niche, has a history of innovations that did not receive market acceptance.

A 2007 Gartner Group paper predicted the following technologies could be disruptive to the business intelligence market.

• Service Oriented Architecture
• Search capabilities integrated into reporting and analysis technology

- Software as a Service
- Analytic tools that work in memory
- Visualization

Another prediction is that data warehouse performance will continue to be improved by use of data warehouse appliances, many of which incorporate the developments in the aforementioned Gartner Group report.

Finally, management consultant Thomas Davenport, among others, predicts that more organizations will seek to differentiate themselves by using analytics enabled by data warehouses.

4.0 CONCLUSION

Data warehouse is now emerging as very important in database management systems. This is as a result the growth in the database of large corporations. A data warehouse now makes it easier for the holding of data while in use. However, there are challenges are constraints in the acceptance and implementation of data warehouse, which is a normal in the development of any concept. The future of data warehouse is good as some organizations will opt for it.

5.0 SUMMARY

- A data warehouse is a repository of an organization's electronically stored data. Data warehouses are designed to facilitate reporting and analysis.
- The concept of data warehousing dates back to the late-1980s when IBM researchers Barry Devlin and Paul Murphy developed the "business data warehouse".
- Architecture, in the context of an organization's data warehousing efforts, is a conceptualization of how the data warehouse is built.
- There are two leading approaches to storing data in a data warehouse - the dimensional approach and the normalized approach.
- Another important decision in designing a data warehouse is which data to conform and how to conform the data.
- Ralph Kimball, a well-known author on data warehousing, is a proponent of the bottom-up approach to data warehouse design.
- Operational systems are optimized for preservation of data integrity and speed of recording of business transactions through use of database normalization and an entity-relationship model.
- Organizations generally start off with relatively simple use of data warehousing. Over time, more sophisticated use of data warehousing evolves.

• A data warehouse appliance is an integrated set of servers, storage, OS, DBMS and software specifically pre-installed and pre-optimized for data warehousing
• Data warehousing, like any technology niche, has a history of innovations that did not receive market acceptance.

6.0 Tutor-Marked Assignment

1. Discuss the benefits associated with the use of data warehouse..
2. Mention 5 applications of data warehouse appliances

7.0 REFERENCES/FURTHER READINGS

Inmon, W.H. Tech Topic: What is a Data Warehouse? Prism Solutions. Volume 1. 1995.

Yang, Jun. WareHouse Information Prototype at Stanford (WHIPS). Stanford University. July 7, 1998.

Caldeira, C. "Data Warehousing - Conceitos e Modelos". Edições Sílabo. 2008. ISBN 978-972-618-479-9

Kimball, R. and Ross, M. "The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling". pp. 310. Wiley. 2nd Ed. 2002. ISBN 0-471-20024-7.

Ericsson, R. "Building Business Intelligence Applications with .NET". 1st Ed. Charles River Media. February 2004. pp. 28-29.

Pendse, Nigel and Bange, Carsten "The Missing Next Big Things",

Schlegel, Kurt "Emerging Technologies Could Prove Disruptive to the Business Intelligence Market", Gartner Group. July 6, 2007

Davenport, Thomas and Harris, Jeanne "Competing on Analytics: The New Science of Winning". Harvard Business School Press. 2007. ISBN 1-422-10332-3.

Queries from Hell blog » When is an appliance not an appliance?

DBMS2 — DataBase Management System Services»Blog Archive » Data warehouse appliances – fact and fiction

Todd White (November 5 1990). "Teradata Corp. suffers first quarterly Loss in four years". Los Angeles Business Journal.

UNIT 3 DOCUMENT MANAGEMENT SYSTEM

CONTENTS

1.0 INTRODUCTION

A document management system (DMS) is a computer system (or set of computer programs) used to track and store electronic documents and/or images of paper documents. The term has some overlap with the concepts of Content Management Systems and is often viewed as a component of Enterprise Content Management Systems and related to Digital Asset Management, Document imaging, Workflow systems and Records Management systems. Contract Management and Contract Lifecycle Management (CLM) can be viewed as either components or implementations of ECM.

2.0 OBJECTIVES

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

- define document management system
- trace the history and development process of document management system
- compare and contrast document management system and content management systems
- know the basic components of document management systems
- answer the question of issues addressed by document management systems
- know the types of document management systems available off the shelf.

3.0 MAIN CONTENT

3.1 History

Beginning in the 1980s, a number of vendors began developing systems to manage paper-based documents. These systems managed paper documents, which included not only printed and published documents, but also photos, prints, etc.

Later, a second system was developed, to manage electronic documents, i.e., all those documents, or files, created on computers, and often stored on local user file systems. The earliest electronic document management (EDM) systems were either developed to manage proprietary file types, or a limited number of file formats. Many of these systems were later referred to as document imaging systems, because the main capabilities were capture, storage, indexing and retrieval of image file formats. These systems enabled an organization to capture faxes and forms, save copies of the documents as images, and store the image files in the repository for security and quick retrieval (retrieval was possible because the system handled the extraction of the text from the document as it was captured, and the text indexer provided text retrieval capabilities).

EDM systems evolved to where the system was able to manage any type of file format that could be stored on the network. The applications grew to encompass electronic documents, collaboration tools, security, and auditing capabilities.

3.2 Document Management and Content Management

There is considerable confusion in the market between document management systems (DMS) and content management systems (CMS). This has not been helped by the vendors, who are keen to market their products as widely as possible.

These two types of systems are very different, and serve complementary needs. While there is an ongoing move to merge the two together (a positive step), it is important to understand when each system is appropriate.

Document Management Systems (DMS)

Document management is certainly the older discipline, born out of the need to manage huge numbers of documents in organisations.

Mature and well-tested, document management systems can be characterised as follows:

- focused on managing documents, in the traditional sense (like Word files)
- each unit of information (document) is fairly large, and self-contained
- there are few (if any) links between documents
- provides limited integration with repository (check-in, check-out, etc)
- focused primarily on storage and archiving
- includes powerful workflow
- targeted at storing and presenting documents in their native format
- limited web publishing engine typically produces one page for each document

Note that this is just a generalised description of a DMS, with most systems offering a range of unique features and capabilities. Nonetheless, this does provide a representative outline of common DMS functionality.

A typical document management scenario:

A large legal firm purchases a DMS to track the huge number of advice documents, contracts and briefs. It allows lawyers to easily retrieve earlier advice, and to use 'precedent' templates to quickly create new documents.

You can't build a website with just a DM system

Content Management Systems (CMS)

Content management is more recent, and is primarily designed to meet the growing needs of the website and intranet markets.

A content management system can be summarised as follows:

- manages small, interconnected units of information (e.g. web pages)
- each unit (page) is defined by its location on the site
- extensive cross-linking between pages
- focused primarily on page creation and editing
- provides tight integration between authoring and the repository (metadata, etc)
- provides a very powerful publishing engine (templates, scripting, etc)

A typical content management scenario:

A CMS is purchased to manage the 3000 page corporate website. Template-based authoring allows business groups to easily create content, while the publishing system dynamically generates richly-formatted pages.

Content management and document management are complementary, not competing technologies. You must choose an appropriate system if business needs are to be met.

3.3 Components

Document management systems commonly provide storage, versioning, metadata, security, as well as indexing and retrieval capabilities. Here is a description of these components:

Metadata

Metadata is typically stored for each document. Metadata may, for example, include the date the document was stored and the identity of the user storing it. The DMS may also extract metadata from the document automatically or prompt the user to add metadata. Systems also use optical character recognition on scanned images, or perform text extraction on electronic documents. The resulting extracted text can be used to assist users in locating documents by identifying probable keywords or providing for full text search capability, or can be used on its own. Extracted text can also be stored as a component of metadata, stored with the image, or separately as a source for searching document collections.

Integration

Many document management systems attempt to integrate document management directly into other applications, so that users may retrieve existing documents directly from the document management system repository, make changes, and save the changed document back to the repository as a new version, all without leaving the application. Such integration is commonly available for office suites and e-mail or collaboration/groupware software.

Capture

Images of paper documents using scanners or multifunction printers. Optical Character Recognition (OCR) software is often used, whether integrated into the hardware or as stand-alone software, in order to convert digital images into machine readable text.

Indexing

Track electronic documents. Indexing may be as simple as keeping track of unique document identifiers; but often it takes a more complex form, providing classification through the documents' metadata or even through word indexes extracted from the documents' contents. Indexing exists mainly to support retrieval. One area of critical importance for rapid retrieval is the creation of an index topology.

Storage

Store electronic documents. Storage of the documents often includes management of those same documents; where they are stored, for how long, migration of the documents from one storage media to another (Hierarchical storage management) and eventual document destruction.

Retrieval

Retrieve the electronic documents from the storage. Although the notion of retrieving a particular document is simple, retrieval in the electronic context can be quite complex and powerful. Simple retrieval of individual documents can be supported by allowing the user to specify the unique document identifier, and having the system use the basic index (or a non-indexed query on its data store) to retrieve the document. More flexible retrieval allows the user to specify partial search terms involving the document identifier and/or parts of the expected metadata. This would typically return a list of documents which match the user's search terms. Some systems provide the capability to specify a Boolean expression containing multiple keywords or example phrases expected to exist within the documents' contents. The retrieval for this kind of query may be supported by previously-built indexes, or may perform more time-consuming searches through the documents' contents to return a list of the potentially relevant documents. See also Document retrieval.

Distribution Security

Document security is vital in many document management applications. Compliance requirements for certain documents can be quite complex depending on the type of documents. For instance the Health Insurance Portability and Accountability Act (HIPAA) requirements dictate that medical documents have certain security requirements. Some document management systems have a rights management module that allows an administrator to give access to documents based on type to only certain people or groups of people.

Workflow

Workflow is a complex problem and some document management systems have a built in workflow module. There are different types of workflow. Usage depends on the environment the EDMS is applied to. Manual workflow requires a user to view the document and decide who to send it to. Rules-based workflow allows an administrator to create a rule that dictates the flow of the document through an organization: for instance, an invoice passes through an approval process and then is routed to the accounts payable department. Dynamic rules allow for branches to be created in a workflow process. A simple example would be to enter an invoice amount and if the amount is lower than a certain set amount, it follows different routes through the organization.

Collaboration

Collaboration should be inherent in an EDMS. Documents should be capable of being retrieved by an authorized user and worked on. Access should be blocked to other users while work is being performed on the document.

Versioning

Versioning is a process by which documents are checked in or out of the document management system, allowing users to retrieve previous versions and to continue work from a selected point. Versioning is useful for documents that change over time and require updating, but it may be necessary to go back to a previous copy.

3.4 Issues Addressed in Document Management

There are several common issues that are involved in managing documents, whether the system is an informal, ad-hoc, paper-based method for one person or if it is a formal, structured, computer enhanced system for many people across multiple offices. Most methods for managing documents address the following areas:

Location	Where will documents be stored? Where will people need to go to access documents? Physical journeys to filing cabinets and file rooms are analogous to the onscreen navigation required to use a document management system.
Filing	How will documents be filed? What methods will be used to organize or index the documents to assist in later retrieval? Document management systems will typically use a database to store filing information.
Retrieval	How will documents be found? Typically, retrieval encompasses both browsing through documents and searching for specific information.
Security How	**will documents be kept secure? How will unauthorized**

	personnel be prevented from reading, modifying or destroying documents?
Disaster Recovery	How can documents be recovered in case of destruction from fires, floods or natural disasters?
Retention period	How long should documents be kept, i.e. retained? As organizations grow and regulations increase, informal guidelines for keeping various types of documents give way to more formal Records Management practices.
Archiving	**How can documents be preserved for future readability?**
Distribution	**How can documents be available to the people that need them?**
Workflow	**If documents need to pass from one person to another, what are the** rules for how their work should flow?
Creation	How are documents created? This question becomes important when multiple people need to collaborate, and the logistics of version control and authoring arise.
Authentication	**Is there a way to vouch for the authenticity of a document?**

3.5 Using XML in Document and Information Management

The attention paid to XML (Extensible Markup Language), whose 1.0 standard was published February 10, 1998, is impressive. XML has been heralded as the next important internet technology, the next step following HTML, and the natural and worthy companion to the Java programming language itself. Enterprises of all stripes have rapturously embraced XML. An important role for XML is in managing not only documents but also the information components on which documents are based.

Document Management: Organizing Files

Document management as a technology and a discipline has traditionally augmented the capabilities of a computer's file system. By enabling users to characterize their documents, which are usually stored in files, document management systems enable users to store, retrieve, and use their documents more easily and powerfully than they can do within the file system itself.

Long before anyone thought of XML, document management systems were originally developed to help law offices maintain better control over and access to the many documents that legal professionals generate. The basic mechanisms of the first document management systems performed, among others, these simple but powerful tasks:

•Add information about a document to the file that contains the document
•Organize the user-supplied information in a database

•Create information about the relationships between different documents

In essence, document management systems created libraries of documents in a computer system or a network. The document library contained a "card catalog" where the user-supplied information was stored and through which users could find out about the documents and access them. The card catalog was a database that captured information about a document, such as these:

•Author: who wrote or contributed to the document
•Main topics: what subjects are covered in the document
•Origination date: when was it started
•Completion date: when was it finished
•Related documents: what other documents are relevant to this document
•Associated applications: what programs are used to process the document
•Case: to which legal case (or other business process) is the document related

Armed with a database of such information about documents, users could find information in more sensible and intuitive ways than scanning different directories' lists of contents, hoping that a file's name might reveal what the file contained. Many people consider document management systems' first achievement to have created "a file system within the file system."

Soon, document management systems began to provide additional and valuable functionality. By enriching the databases of information about the documents (the metadata), these systems provided these capabilities:

•Version tracking: see how a document evolves over time
•Document sharing: see in what business processes the document is used and re-used
•Electronic review: enable users to add their comments to a document without actually changing the document itself
•Document security: refine the different types of access that different users need to the document
•Publishing management: control the delivery of documents to different publishing process queues
•Workflow integration: associate the different stages of a document's life-cycle with people and projects with schedules

These critical capabilities (among others) of document management systems have proven enormously successful, fueling a multi-billion dollar business.

XML: Managing Document Components

XML and its parent technology, SGML (Standard Generalized Markup Language), provide the foundation for managing not only documents but also the information components of which the documents are composed. This is due to some notable characteristics of XML data.

Documents vs. Files

In XML, documents can be seen independently of files. One document can comprise many files, or one file can contain many documents. This is the distinction between the physical and logical structure of information. XML data is primarily described by its logical structure. In a logical structure, principal interest is placed on what the pieces of information are and how they relate to each other, and secondary interest is placed on the physical items that constitute the information.

Rather than relying on file headers and other system-specific characteristics of a file as the primary means for understanding and managing information, XML relies on the markup in the data itself. A chapter in a document is not a chapter because it resides in a file called chapter1.doc but because the chapter's content is contained in the <chapter> and </chapter> element tags.

Because elements in XML can have attributes, the components of a document can be extensively self-descriptive. For example, in XML you can learn a lot about the chapter without actually reading it if the chapter's markup is rich in attributes, as in <chapter language="English" subject="colonial economics" revision_date="19980623" author="Joan X. Pringle" thesis_advisor="Ramona Winkelhoff">. When the elements carry self-describing metadata with them, systems that understand XML syntax can operate on those elements in useful ways, just like a traditional document management system can. But there is a major difference.

Information vs. Documents

XML markup provides metadata for all components of a document, not merely the object that contains the document itself. This makes the pieces of information that constitute a document just as manageable as the fields of a record in a database. Because XML data follows syntactic rules for well-formedness and proper containment of elements, document management systems that can correctly read and parse XML data can apply the functions of document management system, such as those mentioned above, to any and all information components inside the document.
The focus on information rather than documents from XML offers some

important capabilities:

• Reuse of Information

While standard document management systems do offer some measure of information reuse through file sharing, information management systems based on XML or SGML enable people to share pieces of common information without storing the piece of information in multiple places.

• Information Harvesting

By enabling people to focus on information components that make up documents rather than on the documents themselves, these systems can identify and capture useful information components that have ongoing value "buried" inside documents whose value as documents is limited. That is, a particular document may be useful only for a short time, but chunks of information inside that document may be reusable and valuable for a longer period.

• Fine-Granularity Text-Management Applications

Because the information components in XML documents are identifiable, manipulatable, and manageable, XML information management technology can support real economies in applications such as translation of technical manuals.

Evaluating Product Offerings

While the general world of document management and information management is moving toward adoption of structured information and use of XML and SGML, some product offerings distinguish themselves by using underlying database management products with native support for object-oriented data. Object-oriented data matches the structure of XML data quite well and database systems that comprehend object-oriented data adapt well to the tasks of managing XML information.

By contrast, other information management products that comprehend XML or SGML data use relational database systems and provide their own object-oriented extensions to those database systems in order to comprehend object-oriented data such as XML or SGML data, and relying on such implementations have also garnered success and respect in the document management marketplace.

3.6 Types of Document Management Systems

- Alfresco (software)
- ColumbiaSoft
- Main//Pyrus DMS
- OpenKM

- Computhink's ViewWise
- Didgah
- Documentum
- DocPoint
- Hummingbird DM
- Interwoven's Worksite
- Infonic Document Management (UK)
- ISIS Papyrus
- KnowledgeTree
- Laserfiche
- Livelink
- O3spaces
- Oracle's Stellent
- Perceptive Software
- Questys Solutions
- Redmap
- Report2Web
- SharePoint
- Saperion
- SAP KM&C SAP Netweaver
- TRIM Context
- Xerox Docushare

4.0 CONCLUSION

Document management systems have added variety to the pool of options available in datase managemnt in corp[orations. Many products are of the shelf for end users to choose from. The use of document management systems has encouraged the concept and drive for paperless ofice and transactions. It is a concept that truly makes the future bight as man tend toward greater efficiency by eliminating use of papers and hard copies of data and information.

5.0 SUMMARY

- A document management system (DMS) is a computer system (or set of computer programs) used to track and store electronic documents and/or images of paper documents
- Beginning in the 1980s, a number of vendors began developing systems to manage paper-based documents. These systems managed paper documents, which included not only printed and published documents, but also photos, prints, etc.
- There is considerable confusion in the market between document management systems (DMS) and content management systems (CMS).
- Document management systems commonly provide storage, versioning, metadata, security, as well as indexing and retrieval capabilities. Here is a description of these components:
- There are several common issues that are involved in managing documents, whether the system is an informal, ad-hoc, paper-based method for one person or if it is a formal, structured, computer enhanced system for many people across multiple offices
- The attention paid to XML (Extensible Markup Language), whose 1.0 standard was published February 10, 1998, is impressive. XML has been heralded as the next important Internet technology, the next

step following HTML, and the natural and worthy companion to the Java programming language itself. Enterprises of all stripes have rapturously embraced XML.

6.0 TUTOR-MARKED ASSIGNMENT

1. List 5 characteristics of a document management system
2. Discuss briefly workflow in the context of it as a component of document management system

7.0 REFERENCES/FURTHER READINGS

BBC -h2g2 guide Shoebox Storage.

James Robertson, Published on 14 February 2003.A

Miles L. Mathieu, Ernest A. Capozzoli (2002). "The Paperless Office: *Accepting Digitized data" (PDF). Troy State University*.

Kevin Craine. "Excerpts from Designing a Document *Strategy" (HTML). Craine Communications Group.*