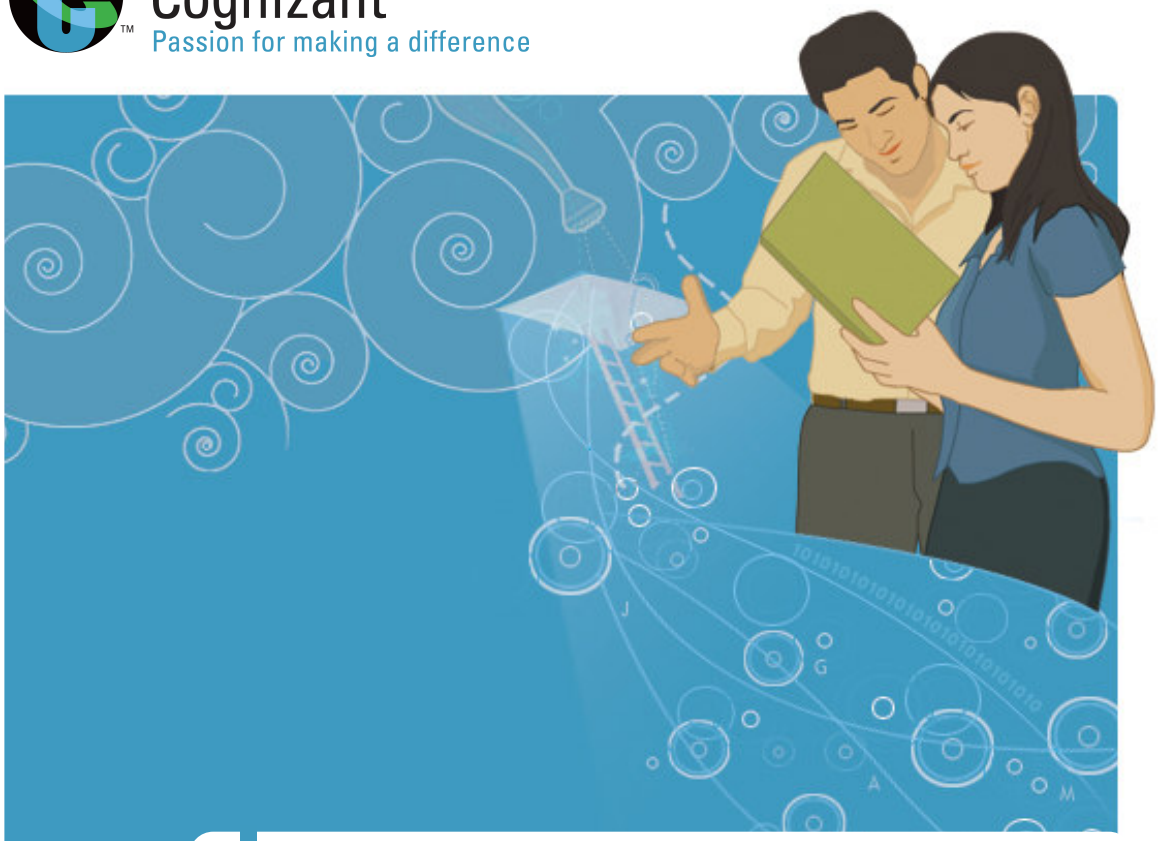
Cognizant
Passion for making a difference



Handout: Database Management System

Version: DBMS/Handout/0307/2.1
Date: 05-03-07

Cognizant
500 Glen Pointe Center West
Teaneck, NJ 07666
Ph: 201-801-0233
www.cognizant.com

TABLE OF CONTENTS

Introduction

About this Document

This document provides the basic concepts of databases. The students can use this as a reference to acquire the skills necessary for designing and working with databases.

Target Audience

In-Campus Trainees

Objectives

- Explain the need of Databases
- Define the different types of Databases
- Explain the Database Design Process using E-R Model and Relational Database Concepts
- Write queries using Structured Query Language
- Describe the Physical File Organisation and File Access Methods
- Explain the Overview of the Database Trends

Pre-requisite

Basic Knowledge of Data Processing and basics of Computer Organisation and Operating System Concepts

Session 1: Introduction to Database Systems

Learning Objectives

After completing this chapter, you will be able to:

- Explain the need for Databases
- Define Database and Database Management Systems
- Describe File-Based Systems and their Limitations
- List the features of Database Systems
- Identify the different database users

Introduction

Today we live in the age where we need data for all our daily life. In the course of a day we interact with different kinds of databases. For instance, a person withdraws money from his bank account on his way to office, then during his office time he checks for some details about some products in the internet and in the evening he purchases grocery items from a nearby supermarket. In this case, the person does nothing but interact with different databases for different needs.

The basic type of database we used is the traditional database which has data in the form of text and numbers. But with the advancement in the Computer and Communication Technology we have new types of databases. Multimedia databases where data includes pictures, video clips and audio along with text; Geographic Information Systems which has data about satellite pictures and weather images; Data warehousing and Online Analytical Processing where some historical data is analyzed using statistical tools which can be used for decision making. Though there are different databases for various applications, it is important to get to the basics of the database concepts. This session gives the basic concepts about the database and database management systems.

A database is a set of related data. By data, we mean known facts that can be recorded and have an implicit meaning. For example, an address book of a person contains the details of his friends and relatives which have an implicit meaning. Now organizations are increasingly aware of the importance of information in the solution of their problems. Because of decreasing cost of data storage, organizations store increasing quantities of data and this data must be managed in the most efficient and effective manner.

File – Based Systems

Earlier organizations were storing data as part of File Systems which is a collection of application programs that perform services for the end users (e.g. reports).In such cases each program defines and manages its own data. Let us consider the case of a real estate agency. It has three departments – sales department, contracts department and data processing department.

The sales department is responsible for selling and renting of properties. Whenever a client approaches the sales department with a view to marketing his or her property , he is required to fill up a form. This gives the details of the property such as address and number of rooms together with the owner's details. The sales department also handles inquiries from clients and a separate form is completed for each one. With the assistance of the data processing department, the sales department creates an information system to handle the renting of property.

The contracts department is responsible for handling the lease agreements associated with properties for rent. Whenever a client agrees to rent a property, a form is filled in by one of the sales staff giving the client and property details. This form is passed to the contracts department which allocates a lease number and completes the payment and rental period details. Again, with the assistance of the data processing department the contracts department creates an information system to handle lease agreements. The system consists of three files storing lease, property and client details, containing similar data held by the sales department.

The situation is illustrated in Fig 1.1. It shows each department accessing their own files through application programs written specially for them. Each set of departmental application programs handles data entry, file maintenance and the generation of a fixed set of specific reports.

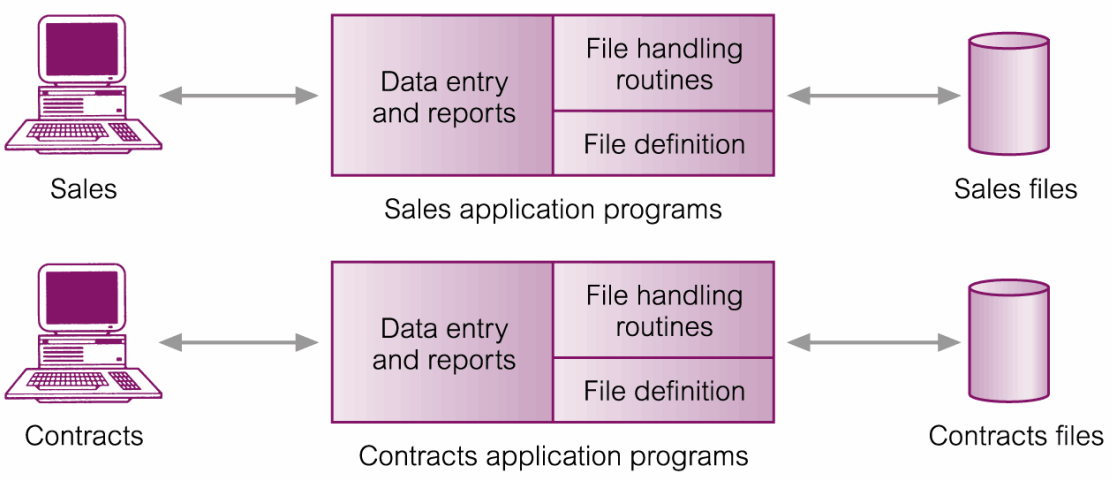


Fig 1.1: File-Based Systems

Limitations of File-Based Approach

Separation and isolation of data

Each program maintains its own set of data. So users of one program may be unaware of potentially useful data held by other programs.

Duplication of data

Same data is held by different programs. Space is wasted. Same item can have potentially different values and/or different formats.

Data dependence

File structure is defined in the program code.

Incompatible file formats

Programs are written in different languages, and so cannot easily access each other's files.

Fixed Queries/Proliferation of application programs

Programs are written to satisfy particular functions. Any new requirement needs a new program.

Database Approach emerged mainly because:

- The definition of data was embedded in application programs, rather than being stored separately and independently.
- There is no control over access and manipulation of data beyond that imposed by application programs.

Thus the definition of a database is:

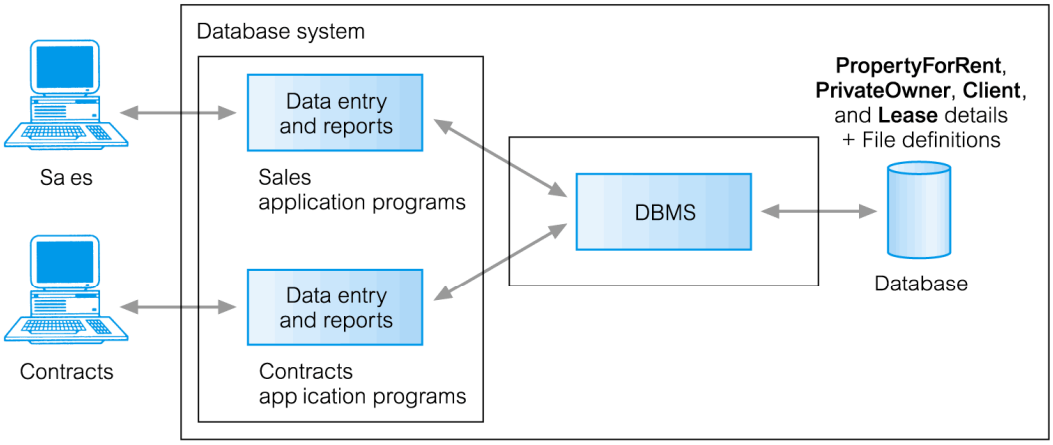
> A database (DB) is a collection of interrelated computer files, whose data contents and structure are described in a data dictionary and which are under the control of a database management system (DBMS)

Database Management Systems

A Database Management System (DBMS) is a software system that enables users to define, create, maintain, and control access to the database -

where defining a database means identify the data in terms of data types, structures and constraints; constructing the database means to load it on a secondary storage medium; manipulating the database means querying, generating reports, insertions, deletions and modifications to its content.

It is assumed that operations (update, insert, retrieve, etc.) on the database can be carried out in a simple and flexible way. Also since a database tends to be a long term resource of an organization, it is expected that planned as well as unplanned applications can (in general) be carried out without great difficulty.



PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType, maxRent)
Lease (leaseNo, propertyNo, clientNo, paymentMethod, deposit, paid, rentStart, rentF nish)

Fig. 1.2: Database Approach

So a Database System consists of a:

- Data definition language (DDL).This permits specification of data types, structures and any data constraints. All specifications are stored in the database.
- Data manipulation language (DML).This contains the general enquiry facility (query language) of the data.

To have a controlled access to the database the dbms includes the following subsystems - a security system,an integrity system,a concurrency control system,a recovery control system,and a user-accessible catalog.

To summaries, a database system consists of :

- The database (data)
- A DBMS (software)
- A DDL and a DML (Part of the DBMS)
- Application programs

Features of Database Systems

Self-describing nature of a database system: A DBMS catalog stores the description of the database. The description is called meta-data). This allows the DBMS software to work with different databases.

Insulation between programs and data: Called program-data independence. Allows changing data storage structures and operations without having to change the DBMS access programs.

Data Abstraction: A data model is used to hide storage details and present the users with a conceptual view of the database.

Support of multiple views of the data: Each user may see a different view of the database, which describes only the data of interest to that user.

Sharing of data and multiuser transaction processing : allowing a set of concurrent users to retrieve and to update the database. Concurrency control within the DBMS guarantees that each transaction is correctly executed or completely aborted. OLTP (Online Transaction Processing) is a major part of database applications.

There other main features of a database system are centralized data management, data independence, and systems integration.

In a database system, the data is managed by the DBMS and all access to the data is through the DBMS providing a key to effective data processing. This contrasts with conventional data processing systems where each application program has direct access to the data it reads or manipulates. In a conventional DP system, an organization is likely to have several files of related data that are processed by several different application programs.

In the conventional data processing application programs, the programs usually are based on a considerable knowledge of data structure and format. In such environment any change of data structure or format would require appropriate changes to the application programs. These changes could be as small as the following:

- Coding of some field is changed. For example, a null value that was coded as -1 is now coded as -9999.
- A new field is added to the records.
- The length of one of the fields is changed. For example, the maximum number of digits in a telephone number field or a postcode field needs to be changed.
- The field on which the file is sorted is changed.

If some major changes were to be made to the data, the application programs may need to be rewritten. In a database system, the database management system provides the interface between the application programs and the data. When changes are made to the data representation, the metadata maintained by the DBMS is changed but the DBMS continues to provide data to application programs in the previously used way. The DBMS handles the task of transformation of data wherever necessary.

This independence between the programs and the data is called *data independence*. Data independence is important because every time some change needs to be made to the data structure, the programs that were being used before the change would continue to work. To provide a high degree of data independence, a DBMS must include a sophisticated metadata management system.

Advantages of DBMS

- Control of data redundancy
- Data consistency
- More information from the same amount of data
- Sharing of data
- Improved data integrity
- Improved security
- Enforcement of standards
- Economy of scale
- Balance conflicting requirements
- Improved data accessibility and responsiveness
- Increased productivity
- Improved maintenance through data independence
- Increased concurrency
- Improved backup and recovery services

Database Users

Users may be divided into those who actually use and control the content (called “Actors on the Scene”) and those who enable the database to be developed and the DBMS software to be designed and implemented (called “Workers Behind the Scene”).

Database administrators: responsible for authorizing access to the database, for co-ordinating and monitoring its use, acquiring software, and hardware resources, controlling its use and monitoring efficiency of operations.

Database Designers: responsible to define the content, the structure, the constraints, and functions or transactions against the database. They must communicate with the end-users and understand their needs.

End-users: they use the data for queries, reports and some of them actually update the database content.

Categories of End-users

Casual: access database occasionally when needed

Naïve:

Naïve or Parametric: they make up a large section of the end-user population. They use previously well-defined functions in the form of “canned transactions” against the database. Examples are bank-tellers or reservation clerks who do this activity for an entire shift of operations.

Sophisticated: these include business analysts, scientists, engineers, others thoroughly familiar with the system capabilities. Many use tools in the form of software packages that work closely with the stored database.

Stand-alone: mostly maintain personal databases using ready-to-use packaged applications. An example is a tax program user that creates his or her own internal database.

Test your Understanding

1. Which of the following is a database administrator's function?
    a. Database design
    b. Backing up the database
    c. Performance monitoring
    d. User coordination
    e. All of the above — Ans. (e)

2. A Computer file can be best described as
    a. A single data item
    b. A general collection of data items
    c. An orderly collection of data items
    d. A random collection of data items
    e. None of these. — Ans.(c)

3. A DBMS is a set of programs that serve as an interface between application programs and a database (T/F) — True

4. A data base is a collection of programs. (T/F) — False

5. The database administrator creates and maintains edit controls regarding changes and additions to the database. (T/F) — True

Session 2: DBMS Architecture

Learning Objectives

After completing this chapter, you will be able to:

- Explain the Three – Level Architecture of DBMS
- Explain the functions of Database Systems
- Describe the Overall System Architecture

DBMS Architecture

Definitions

Entity

An entity is a distinct object (a person, place, thing concept or event) in the organization that is to be represented in the database.

Attribute

An attribute is a property that describes various characteristics of an entity.

Relationship

A relationship is an association between entities.

The major purpose of a database system is to provide users with a view of the system they need and understand. The system hides certain details of how data is stored, created and maintained. Complexity should also be hidden from database users. This is achieved through layering. The three levels are:

- External Level
- Conceptual Level
- Internal Level

Database change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an instance of the database. The overall design of the database is called the database schema.

A DBMS provides a mapping and conversion between these schemas speedily and correctly. The way users perceive the data is called the external level. The way DBMS and the operating system perceive the data is the internal level, where the data is actually stored using the data structures and file organizations. The conceptual level provides both the mapping and the desired independence between the external and internal levels.

The figure below gives an illustration of the three- level architecture of DBMS.

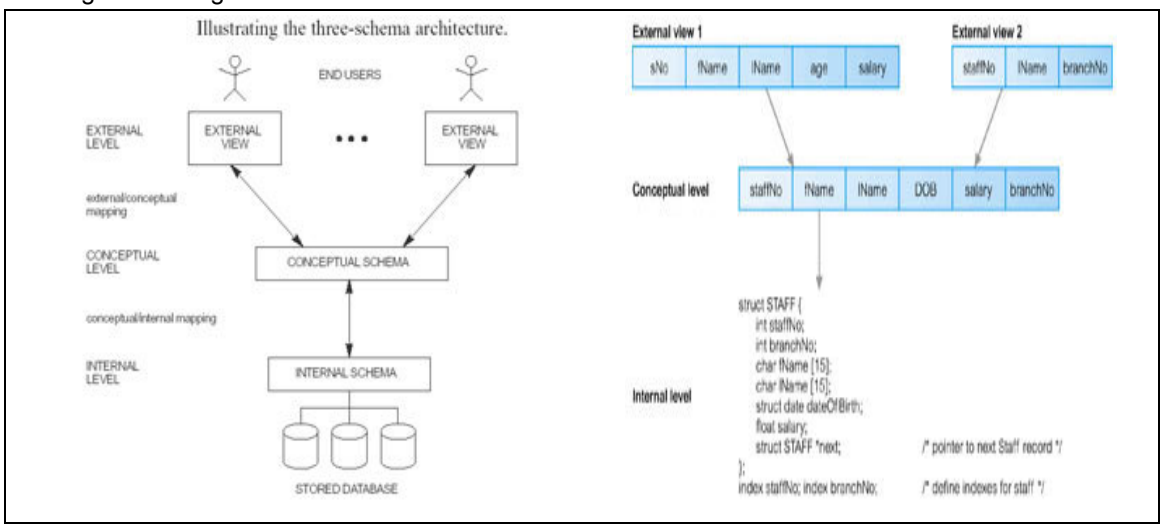


Fig. 2.1: Three Level Architecture

The external level gives the user's view of the database. This level describes that part of the database that is relevant to each user. The external level consists of a number of different external views of the database. Each user has a view of the 'real world' represented in a form that is familiar for that user. The external view includes only those entities, attributes and relationships in the 'real world' that the user is interested. Other entities, attributes and relationships that are not of interest may be represented in the database, but the user will be unaware of them. For example, in a university database, a department head may only be interested in the departmental finances and student enrolments but not the library information. The librarian would not be expected to have any interest in the information about academic staff. The payroll office would have no interest in student enrolments.

The conceptual level gives the community view of the database. This level describes what data is stored in the database and the relationships among the data. This level contains the logical structure of the entire database. It represents all entities, their attributes and their relationships, the constraints on the data, security and integrity information. However, this level should not contain any storage-dependent details. For instance, the description of an entity should contain only data types of attributes (for example, integer, real, character) and their length (such as the maximum number of digits or characters), but not any storage considerations such as the number of bytes occupied.

The internal level gives the physical representation of the database on the computer. This level describes how the data is stored in the database. It covers the data structures and file organizations used to store data on storage devices. It interfaces with the operating system access methods to place the data on the storage devices, build the indexes, and retrieve the data and so on. It is also concerned with data compression and data encryption techniques.

Assuming the three level view of the database, a number of mappings are needed to enable the users working with one of the external views.

For example, the payroll office of XYZ Company may have an external view of the database that consists of the following information only:

Staff number, name and address.

Staff tax information e.g. number of dependents.

Staff employment status, salary level, leave information etc.

The conceptual view of the database may contain academic staff, general staff, casual staff etc. A mapping will need to be created where all the staff in the different categories is combined into one category for the payroll office. The conceptual view would include information about each staff's position, the date employment started, full-time or part-time, etc . This will need to be mapped to the salary level for the salary office. Also, if there is some change in the conceptual view, the external view can stay the same if the mapping is changed.

Data Independence

A major objective for the three-level architecture is to provide data independence, which means that upper levels are unaffected by changes to lower levels. There are two kinds of data independence: **logical** and **physical**.

Logical data independence refers to the immunity of the external schemas to changes in the conceptual schema. Changes to the conceptual schema such as the addition or removal of new entities, attributes or relationships, should be possible without having to change existing external schemas or having to rewrite application programs.

Physical data independence refers to the immunity of the conceptual schema to changes in the internal schema.

Changes to the internal schema, such as using different file orgainsations or storage structures, using different storage devices, modifying indexes, or hashing algorithms, should be possible without having to change the conceptual or external schemas.

The figure below illustrates where each type of data independence occurs in relation to the three-level architecture.

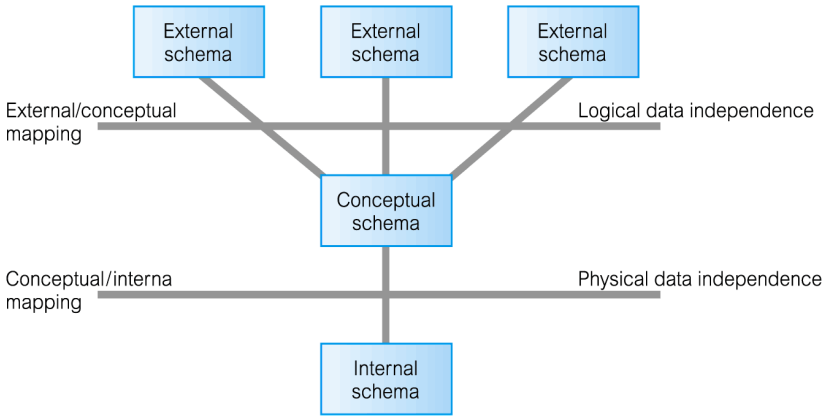


Fig 2.2: Data Independence

Functions of DBMS

Data storage, retrieval and update: A DBMS provides users with the ability to store, retrieve and update data in the database.

A user-accessible catalog: A DBMS provides a catalog in which descriptions of data items are stored and which is accessible to users.

Transaction support: A DBMS provides a mechanism which will ensure either that all the updates corresponding to a given transaction are made or that none of them is made.

Concurrency control services: A DBMS provides a mechanism to ensure that the databse is updated correctly when multiple users are updating the database concurrently.

Recovery services: A DBMS provides a mechanism for recvering the databse in the event that the database is damaged in any way.

Authorization services: A DBMS provides a mechanism to ensure that only authorized users can access the database.

Support for data communication: A DBMS must be capable of integrating with communication software

Integrity services: A DBMS provides a means to ensure that both the data in the database and changes to the data follow certain rules.

Services to promote data independence: A DBMS provides facilities to support the independence of programs from the actual structure of the database.

Database System Structure

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

The storage manager is important because databases typically require a large amount of storage space. The query processor is important because it helps the database system simplify and facilitate access to data. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level into an efficient sequence of operations at the physical level.

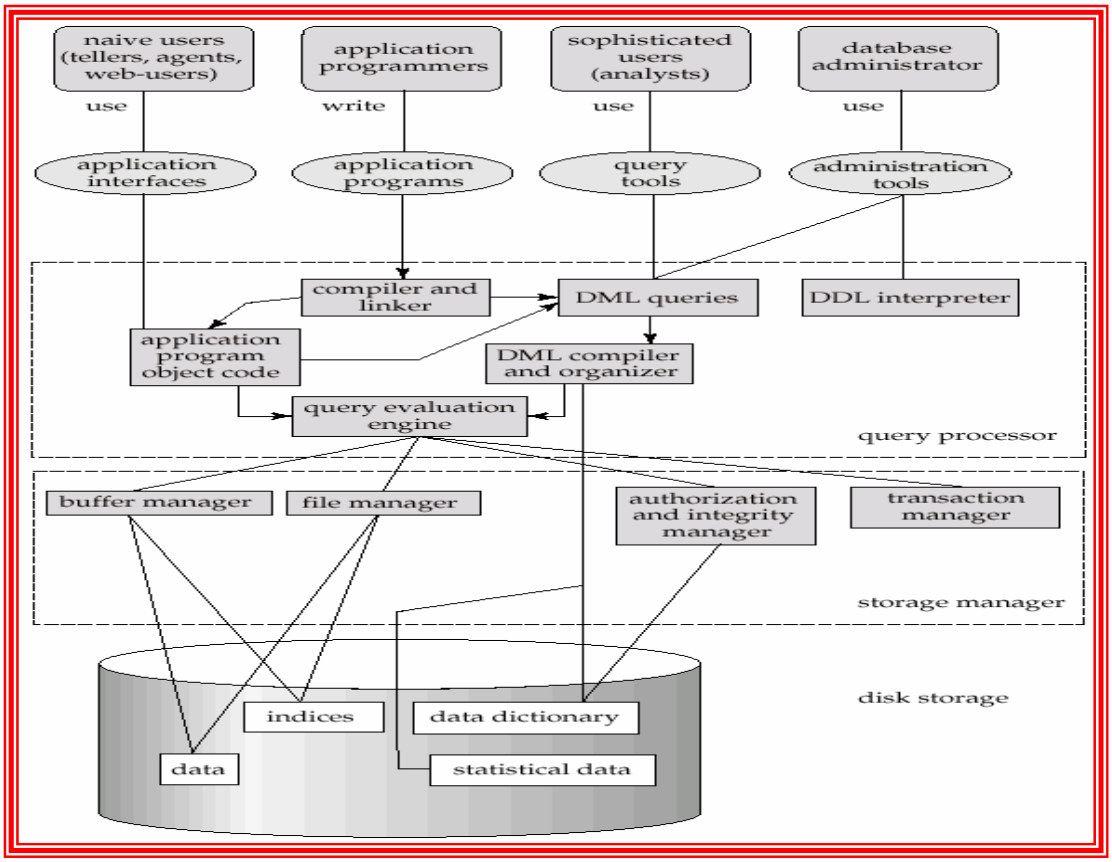


Fig. 2.3: Overall System Architecture

The figure above shows the overall system architecture. The subsystems are transaction manager, query processor and storage manager.

Test your Understanding

1. The Data dictionary tells the DBMS
    a. What files are in the database
    b. What attribute are possessed by the data
    c. what these files contain
    d. All of the above — Ans. (d)

2. The distinguishable parts of a record are called
    a. Files
    b. Data
    c. Fields
    d. Database
    e. None of these. — Ans. (a)

3. Large collections of files are called
    a. Fields
    b. Records
    c. Databases
    d. File system
    e. None of these — Ans. (c)

Session 3: Types of Databases

Learning Objectives

After completing this chapter, you will be able to:

- Define the various types of databases

Hierarchical Databases

The Hierarchical model is the oldest of the database models, and unlike the other data models, does not have a well documented history of its conception and initial release. It is derived from the Information Management Systems (IMS) of IBM in the 1950's and 60's.

It was adopted by many banks and insurance companies who are still running it as legacy systems to this day. Hierarchical database systems can also be found in inventory and accounting systems used by government departments and hospitals.

Structure:

As its name implies, the Hierarchical Database Model defines hierarchically-arranged data.

Perhaps the most intuitive way to visualize this type of relationship is by visualizing an upside down tree of data. In this tree, a single table acts as the "root" of the database from which other tables "branch" out.

Relationships in such a system are thought of in terms of children and parents such that a child may only have one parent but a parent can have multiple children. Parents and children are tied together by links called "pointers" (physical addresses inside the file system). A parent will have a list of pointers to each of their children.

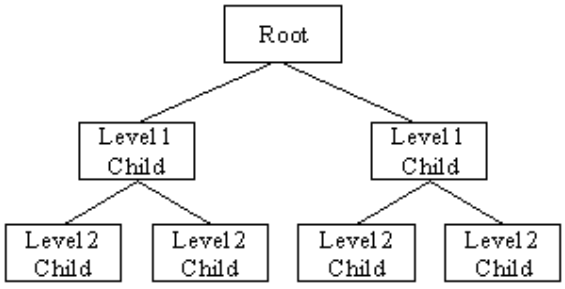


Fig 3.1: Hierarchical Database Model

This child/parent rule assures that data is systematically accessible. To get to a low-level record, you start at the root and work your way down through the tree until you reach your target. Of course, as you might imagine, one problem with this system is that the user must know how the tree is structured in order to find anything.

The hierarchical model however, is much more efficient than the flat-file model because there is not as much need for redundant data. If a change in the data is necessary, the change might only need to be processed once. Consider the student flat file

Table 3.1: Student Flat File

Name	Address	Course	Grade
Mr. Eric Tachibana	123 Kensigton	Chemistry 102	C+
Mr. Eric Tachibana	123 Kensigton	Chinese 3	A
Mr. Eric Tachibana	122 Kensigton	Data Structures	B
Mr. Eric Tachibana	123 Kensigton	English 101	A
Ms. Tonya Lippert	88 West 1st St.	Psychology 101	A
Mrs. Tonya Ducovney	100 Capitol Ln.	Psychology 102	A
Ms. Tonya Lippert	88 West 1st St.	Human Cultures	A
Ms. Tonya Lippert	88 West 1st St.	European Governments	A

As mentioned, this flat file database would store an excessive amount of redundant data. If we implemented this in a hierarchical database model, we would get much less redundant data. Consider the following hierarchical database schema:

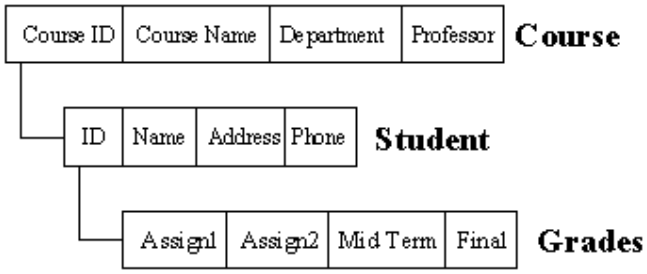


Fig 3.2: Hierarchical Database Schema

In this schema, to access the mid term marks of a student, we have to traverse from the top, ie the course record, identify the course, then move down to student record and then finally we retrieve the mid term marks of the student.

Limitations

However, the hierarchical database model has some serious problems. For one, we cannot add a record to a child until it has already been incorporated into the parent. This might be troublesome if, for example, you wanted to add a student who had not yet signed up for any courses.

The hierarchical database model still creates repetition of data within the database. You might imagine that in the database system shown above, there may be a higher level that includes multiple course. In this case, there could be redundancy because students would be enrolled in several courses and thus each "course tree" would have redundant student information.

Redundancy would occur because hierarchical databases handle one-to-many relationships well but do not handle many-to-many relationships well. This is because a child may only have one parent. However, in many cases you will want to have the child be related to more than one parent. For instance, the relationship between student and class is a "many-to-many". Not only can a student take many subjects but a subject may also be taken by many students. How would you model this relationship simply and efficiently using a hierarchical database? The answer is that it is possible but it is very complex. It can be done by having intermediate connecting nodes called the Virtual Parent-Child relationship.

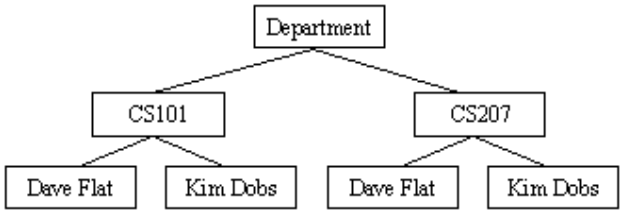


Fig 3.3 Example of Hierarchical Database Model

Faced with these serious problems, the database community came up with the network model.

Network Databases

The Network database model was first introduced in 1971 by CODASYL Data Base Task Group and because of this is sometimes called the DBTG Model. The concept of the network model comes with the idea of IDS (Integrated Data Store) database.

It is called the Network Model because it represents the data it contains in the form of a network of records and sets which are related to each other, forming a network of links.

Key Concepts:

Records are sets of related data values. They store the name of the record type, the attributes associated with it and the format for these attributes

Student				
Surname	Name	Address	Course	Year

data item name	format
Surname	character 32
Name	character 32
Address	character 128
Course	character 30
Year	Int

Fig 3.4: Network Data Model

Record Types, are set of records of same type. These are the equivalent of tables in the relational model.

Set Types, are named, 1:N relationships between 2 record types. These do not have a direct parallel in the relational model, the closest comparison is to a query statement which has joined two tables together. This makes the network model faster to run certain queries. An example of a set type would been the relationship between a university department and the students in it. The network model uses a Bachman diagram to represent these relationships as we can see below,

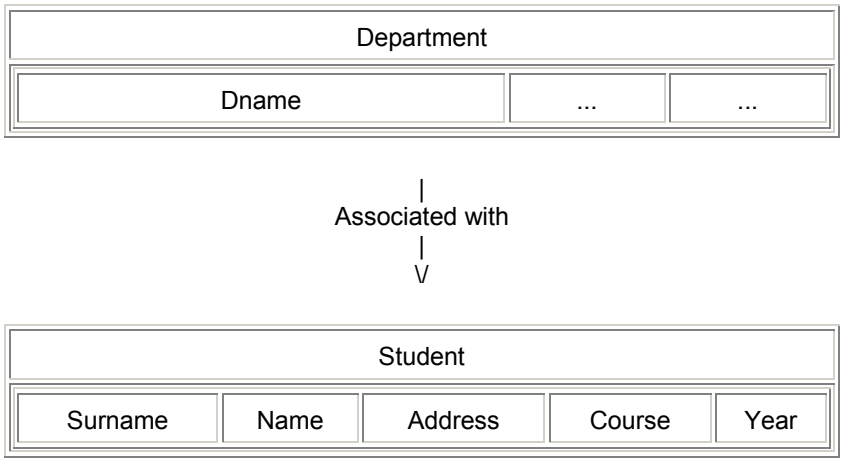


Fig 3.5: Bachman diagram Representation

This relationship is called "Associated with" and is used to set the relationship between the departments in the university, and the individual students in them.

The network model is not commonly used today to design database systems, however, there are a few instances of it being used by companies as part of a legacy system. Because it represents data in a format closer to the way in which it stores it than the other models, it can be faster than relational systems. Unfortunately, this increase in speed is at the cost of its adaptability.

Relational Databases

The Relational Model is the data model which is based on the relational algebra of mathematics. It was first outlined in a paper published by Ted Codd in 1970. It is used by the majority of business community today.

Structure:

The relational model is based on Set Theory and Predicate Logic. Relational databases try to hide as much of the implementation detail from their users as possible.

The use of set theory allows data to be structured in a series of tables, which have columns representing the variables and rows that contain specific instances of data. These tables are organized using Normal Forms.

For example - in a university environment we may have a table with columns labeled surname, name, address, course, year etc. and rows with individual students information. E.g.

Table 3.2: Relational Data Model

LName	FName	Address	Course	Year
Doe	John	Albert Einstein Hall	Computer Science	4
Jones	Paul	Marie Curie Hall	Maths	3
Smith	Jane	Marie Curie Hall	Physics	2

Operations, which can be carried out on the data, include insert, query and delete commands.

Query operations are the most interesting as there are often several ways to obtain the same information, but with different processing time requirements to gather the information.

```
Select name, surname
from students
where address='Albert Einstein Hall';
```

Users can also query several tables at the same time using a **join** command.

The **join** command is used to merge the data in two or more tables together according to a common value. For example, in one table, which we will call *employees*, we can store an employees details.

Table 3.3: Example for a JOIN

Empid	Surname	Name	Dept
100101	Williamson	Peter	IT
100102	Doe	John	EP

Object Oriented Databases

A traditional database stores just data – with no procedures. A traditional relational database does not store procedures, but in addition it attempts to make the data it stores completely Independent form the procedures that access it. Diverse users make the data accessible for diverse purposes through the use of relational database software.

In contrast, an Object-oriented database (OODB) stores objects. An object consists of data accompanied by methods, which implement the actual procedures that process the data.

The following are the few terms associated with object technology:

Object Type

Is a type of entity in the real world, either real or abstract about which we store data Includes a specification of the types of operations that can be performed to manipulate the data stored about that object type.

Operation

Is an activity that is performed to access or manipulate the data associated with an object type. Operations associated with an object type reference only the data associated with that object type and not the data in any other object type

Method

- It specifies the way in which an operation that is performed on the data associated with an object type is encoded in computer software.
- It consists of procedures that can be executed in a computer to access the data associated with an object type.

Class

- It is a computer implementation of an object type
- It includes the data structure that defines the data associated with the object type and a set of methods specifying all the operations that can be performed on the data associated with that class.

Object

- An object is an instance of an object type.
- Consists of a description of the set of data items associated with the object instance
- Consists of a description of the computer code that implements the methods associated with the object's type.

Encapsulation

- Refers to the way in which the object's data and the operations that can be performed on that data are packaged together.
- It hides the implementation details of an object from its users

Inheritance

- Refers to the characteristic of an object type that allows it to take on (inherit) the properties of its parent class including the data structures and the methods used by the parent class
- An object type can override an inherited property with data structures and methods of its own.

Test your Understanding

1. The hierarchical database stores relationship as
   a. Set Type
   b. Parent – Child Relationship type
   c. Primary key
   d. None of the above

   Ans. (b)

2. The Network Database represents M:N relationship
   a. True
   b. False

   Ans. (b)

3. Rows are called as _____ in a Relational Database
   a. Fields
   b. Records
   c. Databases
   d. Tuples

   Ans. (d)

Session 4: Introduction to Data Modeling

Learning Objectives

After completing this chapter, you will be able to:

- List the steps in Database Design Process.
- Design Conceptual Database using ER Model
- Explain the representations in a E-R Diagram

Database Design Process

An Information system includes the resources that enable collection, management, control, and dissemination of information throughout an organization. Since Database is a fundamental component of IS, its development/usage should be viewed from perspective of the wider requirements of the organization. Thus database design process comes as a part of Information System Analysis and Design.

The database design process starts with identification of that part of the real world which needs to be converted into an Information System. Database planning should include development of standards that govern:

- How data will be collected,
- How the format should be specified,
- What necessary documentation will be needed,
- How design and implementation should proceed.

Requirements Collection and Analysis phase is the Process of collecting and analyzing information about the part of organization to be supported by the database system, and using this information to identify users' requirements of new system. Information is gathered for each major user view including:

- A description of data used or generated;
- Details of how data is to be used/generated;
- Any additional requirements for new database system.

Data model representing single user view (or a subset of all user views) is called a local data model.

Each model includes diagrams and documentation describing requirements for one or more but not all user views of database.

Database Design phase refers to the process of creating a design for a database that will support the enterprise's mission statement of the enterprise and mission objectives for the required database system.

Main approaches include:

- Top-down
- Bottom-up
- Inside-out
- Mixed

A data model ensures we understand:

- Each user's perspective of the data;
- Nature of the data itself, independent of its physical representations;
- Use of data across user views.

Three phases of database design includes:

- Conceptual database design (uses E-R Model)
- Logical database design (Relational Data Model)
- Physical database design.

Conceptual Database Design refers to the process of constructing a model of the data used in an enterprise, independent of all physical considerations. Data model is built using the information in users' requirements specification. Conceptual data model is the source of information for logical design phase.

Logical Database Design refers to the process of constructing a model of the data used in an enterprise based on a specific data model (e.g. relational), but independent of a particular DBMS and other physical considerations.

Conceptual data model is refined and mapped on to a logical data model.

Physical Database Design refers to the process of producing a description of the database implementation on secondary storage. It describes base relations, file organizations, and indexes used to achieve efficient access to data. Also describes any associated integrity constraints and security measures. Tailored to a specific DBMS system.

Entity – Relationship Model (E –R Model)

A database can be modeled as:

- A collection of entities,
- Relationship among entities.

An entity is an object that exists and is distinguishable from other objects.

For example:
Specific person, company, event, plant

An entity set is a set of entities of the same type that share the same properties.

For example: Set of all persons, companies, trees, holidays

An entity is represented by a set of attributes that is descriptive properties possessed by all members of an entity set.

Example: customer=(customer-name, social-security, customer-street, customer-city) account = (account-number, balance)

Attribute

- An attribute describes the property of an entity or a relationship type.

Attribute Domain

- Set of allowable values for one or more attributes.

Types of Attribute

- *Simple* and composite attributes.
- *Single-valued* and *multi-valued* attributes.
- *Null* attributes.
- *Derived* attributes.

Simple and composite attributes.

- Simple Attribute: Attribute composed of a single component with an independent existence.
- Composite Attribute: Attribute composed of multiple components, each with an independent existence.

Single-valued and multi-valued attributes.

- Single-valued Attribute: Attribute that holds a single value for each occurrence of an entity type.
- Multi-valued Attribute: Attribute that holds multiple values for each occurrence of an entity type.

NULL Attribute:

Represents value for an attribute that is currently unknown or not applicable for tuple.

Derived Attribute

Represents a value that is derivable from value of a related attribute, or set of attributes, not necessarily in the same entity type.

Relational Keys

- Superkey
  a. An attribute or a set of attributes that is used to uniquely identify a tuple within a relation.
- Candidate Key
  b. Superkey (K) such that no proper subset is a superkey within the relation.
  c. In each tuple of R, values of K uniquely identify that tuple (uniqueness).
  d. No proper subset of K has the uniqueness property (irreducibility).
- Primary Key
  e. Candidate key selected to identify tuples uniquely within relation.
- Alternate Keys
  f. Candidate keys that are not selected to be primary key.
- Foreign Key
  g. Attribute, or set of attributes, within one relation that matches candidate key of some (possibly same) relation.

The Fig 4.1 illustrates an Entity-Relationship Diagram with all the entities, attributes and relationships.

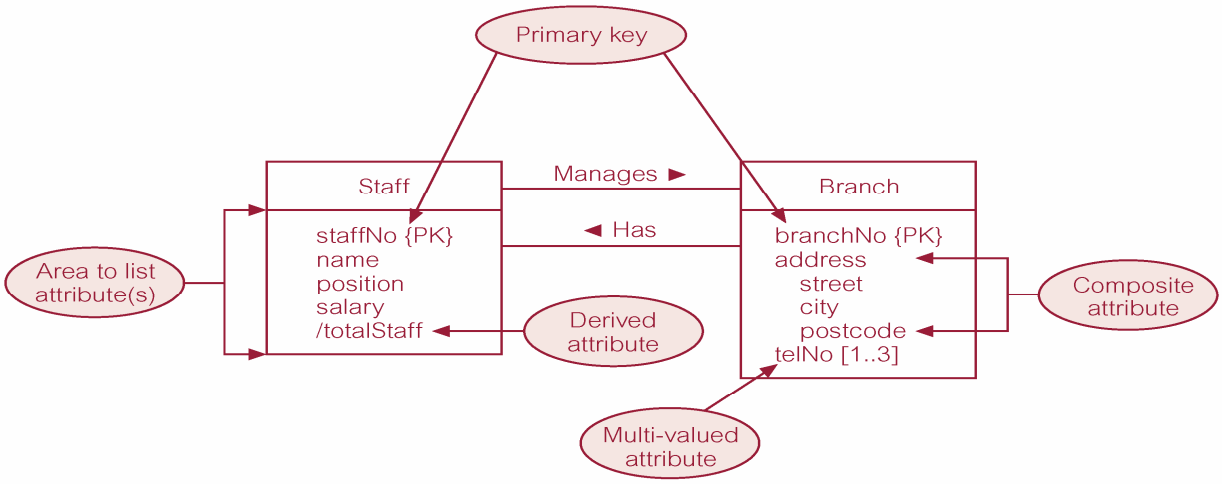


Fig 4.1: ER diagram of Staff and Branch entities and their attributes

Instances of Branch and Staff Relations

E-R Diagram Components

- **Rectangles** represent entity sets.
- **Ellipses** represent attributes
- **Diamonds** represent relationship sets.
- **Lines** link attributes to entity sets and entity sets to relationship sets.
- **Double ellipses** represent multivalued attributes.
- **Dashed ellipses** denote derived attributes.
- Primary key attributes are underlined.

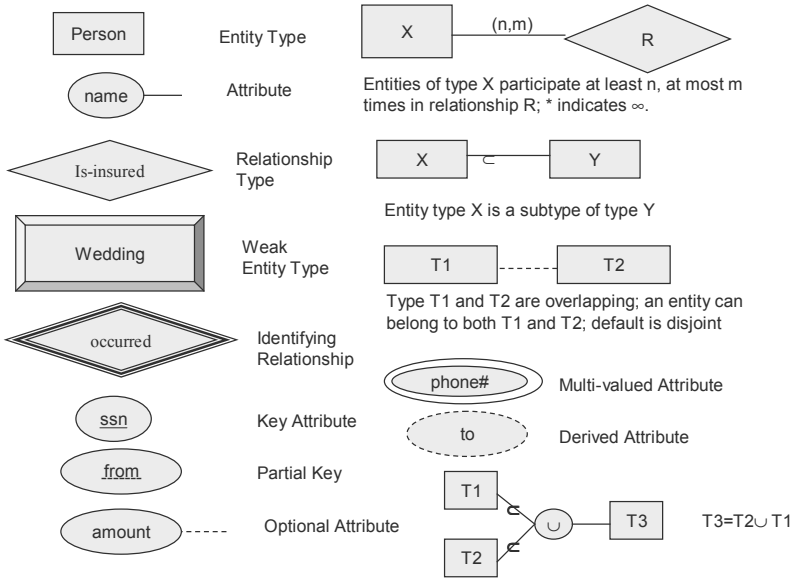


Fig 4.2: E-R Diagram Components

Structural Constraints

Cardinality: Describes maximum number of possible relationship occurrences for an entity participating in a given relationship type.

Participation: Determines whether all or only some entity occurrences participate in a relationship.

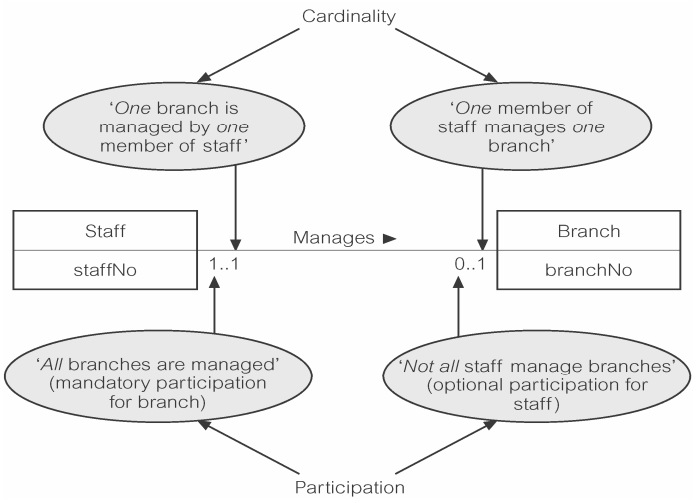


Fig 4.3: Cardinality and Participation

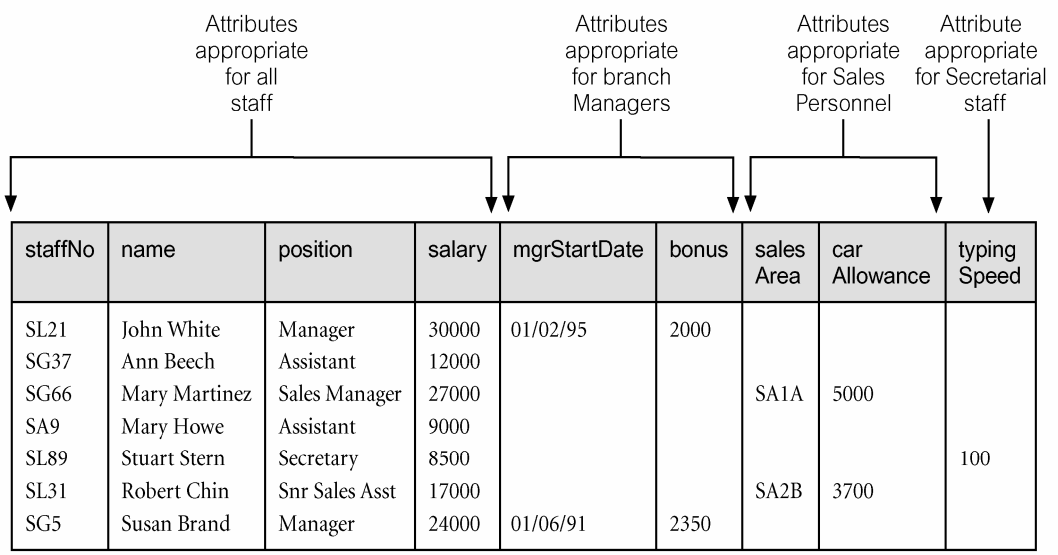
Specialization / Generalization

Associated with special types of entities known as superclasses and subclasses, and the process of attribute inheritance

- Superclass
  h. An entity type that includes one or more distinct subgroupings of its occurrences.
- Subclass
  i. A distinct subgrouping of occurrences of an entity type.
- Superclass/subclass relationship is one-to-one (1:1). Superclass may contain overlapping or distinct subclasses. Not all members of a superclass need be a member of a subclass.
- Attribute Inheritance
  j. An entity in a subclass represents same 'real world' object as in superclass, and may possess subclass-specific attributes, as well as those associated with the superclass.

Specialization: Process of maximizing differences between members of an entity by identifying their distinguishing characteristics.

Generalization: Process of minimizing differences between entities by identifying their common characteristics.



staffNo	name	position	salary	mgrStartDate	bonus	sales Area	car Allowance	typing Speed
SL21	John White	Manager	30000	01/02/95	2000			
SG37	Ann Beech	Assistant	12000					
SG66	Mary Martinez	Sales Manager	27000			SA1A	5000	
SA9	Mary Howe	Assistant	9000					
SL89	Stuart Stern	Secretary	8500					100
SL31	Robert Chin	Snr Sales Asst	17000			SA2B	3700	
SG5	Susan Brand	Manager	24000	01/06/91	2350			

Fig 4.4: All Staff relation holding details of all staff

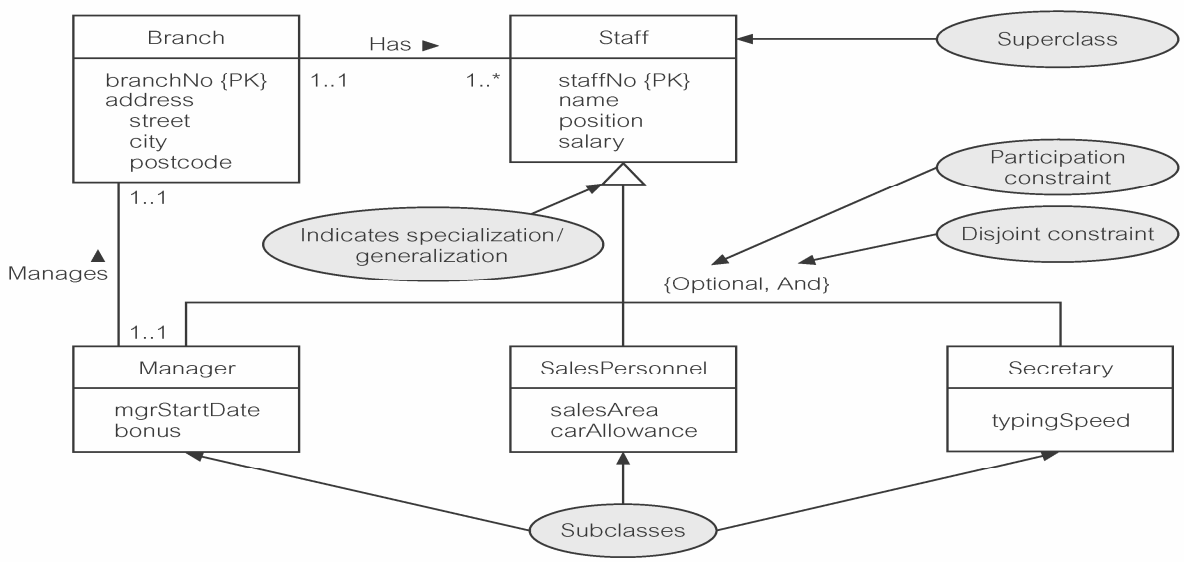


Fig 4.5: Specialization/generalization of Staff entity into subclasses representing job roles

Constraints on Specialization / Generalization

Two constraints that may apply to a specialization/generalization:

- Participation constraints & disjoint constraints.

Participation constraint :

- Determines whether every member in superclass must participate as a member of a subclass.
- May be *mandatory* or *optional*.

Disjoint constraint:

- Describes relationship between members of the subclasses and indicates whether member of a superclass can be a member of one, or more than one, subclass.
- May be *disjoint* or *nondisjoint*.

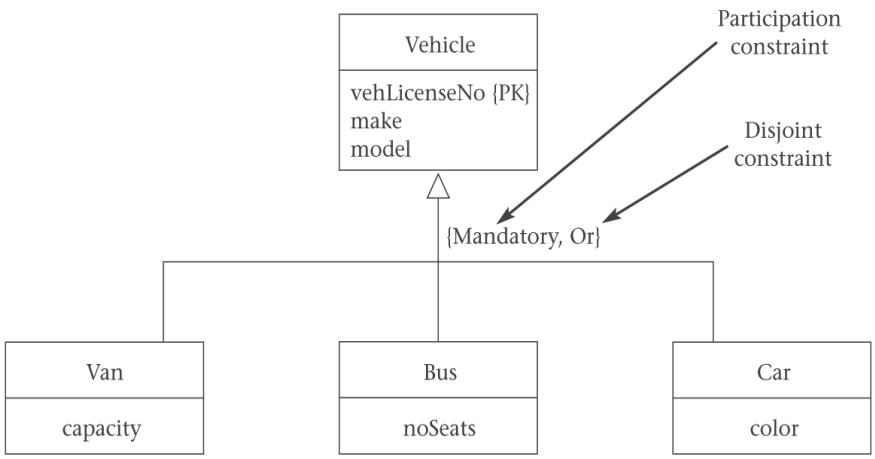


Fig 4.6: Vehicle entity into vehicle types

There are four categories of constraints of specialization and generalization:

- Mandatory and disjoint
- Optional and disjoint
- Mandatory and nondisjoint
- Optional and nondisjoint.

Participation constraint	Disjoint constraint	Tables required
Mandatory	Nondisjoint {And}	Single table
Optional	Nondisjoint {And}	Two tables: one table for superclass and one table for all subclasses
Mandatory	Disjoint {Or}	Many tables: one table for each combined superclass/subclass
Optional	Disjoint {Or}	Many tables: one table for superclass and one table for each subclass

Fig 4.7: Creating tables to represent specialization/generalization

Mapping Cardinalities

- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
  k. One to one

l. One to many
m. Many to one
n. Many to many

- We distinguish among these types by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set.

One-To-One Relationship

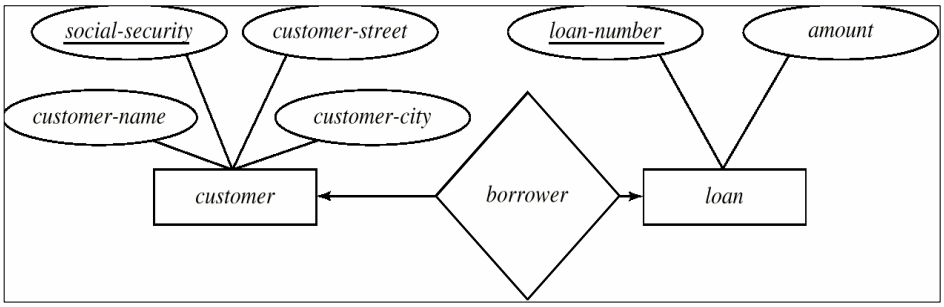


Fig. 4.7: One-To-One Relationship

- A customer is associated with at most one loan via the relationship borrower
- A loan is associated with at most one customer via borrower

One-To-Many and Many-To-One Relationships

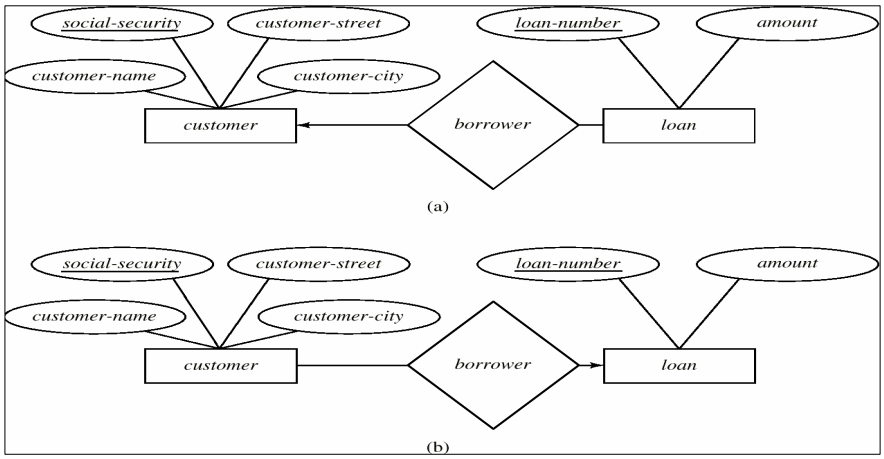


Fig. 4.8: (a) One-To-Many (b) Many-To-One Relationship

- In the one-to-many relationship (a), a loan is associated with at most one customer via *borrower*, a customer is associated with several (including 0) loans via borrower
- In the many-to-one relationship (b), a loan is associated with several (including 0) customers via *borrower*, a customer is associated with at most one loan via *borrower*

Many-To-Many Relationship

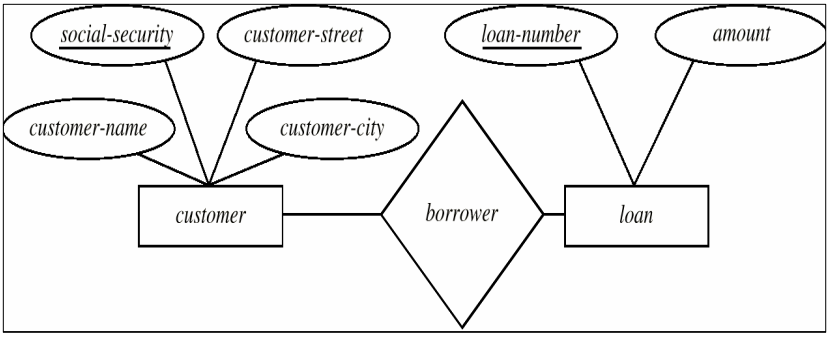


Fig. 4.9: Many-To-Many Relationship

- A customer is associated with several (possibly 0) loans via borrower
- A loan is associated with several (possibly 0) customers via borrower

Test your Understanding

Draw an entity-relationship diagram to illustrate the case given below:

In a certain educational institution, a student can register for several courses, for instance, a course in Calculus, Algebra, or Spanish History. A student has to register for at least one course. Every course offered has a code and a title, and assigned to a classroom, a day of the week, and a time at which it is conducted. Only one instructor teaches a course. However, an instructor can teach several courses.

A student can be a day scholar or a resident scholar. A resident scholar is assigned to a hostel, and a hostel room. Day scholars are assigned lockers to store their belongings while on campus.

During the course, the student is required to appear in several examinations. A course must have at least one examination; a course can have several examinations. The examinations may be of many types – tests, quizzes, seminars and projects. The date, time and place of these examinations are assigned at the beginning of the course. The instructors evaluate the examinations. Several instructors may correct the same set of answer scripts belonging to a course. An instructor may have to correct the answer scripts for more than one examination. An instructor may not correct any answer scripts. A record of the student's marks has to be maintained.

Session 5: E-R Model

Learning Objectives

After completing this chapter, you will be able to:

- Explain Relational Data Model
- Implement Relational Operators in your queries
- Convert E-R Model to Tables

Relational Data Model

In a Relational Model, the data is logically perceived as tables. Tables are logical data structures that we assume hold the data that the database intends to represent. Tables are not physical structures. Each table has a unique name. Tables consist of a given number of columns or attributes. Every column of a table must have a name. No two columns of the same table may have identical names. The total number of columns or attributes that comprises a table is known as the degree of the table.

Tables are required to have at least one column. Tables are not required to have rows. A table with no rows is called an empty table. The process of inserting tuples for the very first time into a table is called populating the table.

For each column of a table there is a set of possible values called its domain. The domain contains all permissible values that can appear under that column. The order of the rows is irrelevant since they are identified by their content and not by their position within the table. No two rows or tuples are identical to each other in all their entries.

The notion of keys is the fundamental concept in the relational model. It provides the basic mechanism for retrieving tuples within any table of the database. To distinguish a candidate key,no two different tuples of the relation will have identical entries in all attributes of the key. The number of attributes that comprises the key must be minimal

Since a relation may have more than one candidate key, one of these candidate keys should be designated as the primary key (PK) of the relation. The values of the primary key can be used as the identification and the addressing mechanism of the relation. Once a primary key has been selected, the remaining candidate keys, if they exist are called alternate keys.

A Relational Model allows only one primary key per table. A primary key may be composed of a single attribute (single primary key) or more than one attribute (composite primary key) .Attributes that are part of any key are called prime attributes. Since the primary key is used to identify the tuples or rows of a relation, none of its attributes may be NULL. In a relation, the NULL value is used to represent missing information, unknown, or inapplicable data. A NULL value is not a zero value A NULL value doesn't represent a particular value within the computer. This imposes an additional condition or constraint on the keys known as the integrity constraint.

Because columns that have the same underlying domain can be used to relate tables of a database, the concept of foreign key (FK) allows the DBMS to maintain consistency among the rows of two relations or between the rows of the same relation. The attributes of a FK have the same underlying domain as the set of attributes defined for the PK.

The FK values in any tuple of a relation are either NULL or must appear as the PK values of a tuple of the corresponding relation.

Alternative Terminology for Relational Model

Formal terms	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

Relational Operators

Relational Model is based on the relational algebra of mathematics .Relational operators are part of the relational algebra which are mainly used to retrieve data from the tables.

There are 8 relational operators

1. Restrict
2. Project
3. Product
4. Union
5. Intersect
6. Difference
7. Join
8. Dividion

These operators take one or more relations as inputs and give a new relation as a result.

Restrict

Works on a single relation R and defines a relation that contains only those tuples (rows) of R that satisfy the specified condition (predicate).

Project

Projection

Works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.

Product

Works on two or more relations and produces a Cartesian product of the relations as the result.

Union

Union of two relations R and S defines a relation that contains all the tuples of R, or S, or both R and S, duplicate tuples being eliminated. To be union-compatible both the tables should have the same set of attributes. R and S must be union-compatible.

Intersect

Intersection

Defines a relation consisting of the set of all tuples that are in both R and S.

R and S must be union-compatible.

Difference

Works on two relations and retrieves data in the first relation which is not in the second relation.

Defines a relation consisting of the tuples that are in relation R, but not in S.

R and S must be union-compatible.

Product

Defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.

Join Operations

Join is a derivative of Cartesian product. Equivalent to performing a Selection, using join predicate as selection formula, over Cartesian product of the two operand relations.

Division

Defines a relation over the attributes C that consists of set of tuples from R that match combination of every tuple in S.

Converting ER Diagram to Relational Database Design

Converting Strong Entity Types

- Each entity type becomes a table
- Each single valued attribute becomes a column
- Derived attributes are ignored
- Composite attributes are represented by components
- Multi valued attributes are represented by a separate table
- The key attribute of the entity type becomes the primary key of the table

Converting Weak Entity Types

It is converted to a table of their own, with the primary key of the strong entity acting as a foreign key in the table. This foreign key along with the key of the weak entity form the composite primary key of this table.

Converting Relationships

The relationships representation is based on cardinality and degree

Cardinalities - 1:1, 1:M, M:N

Unary 1:1

Example: Employee who are also a couple

The primary key itself will become the foreign key in the same table

Unary 1:N

Example: Employee who is also a manager

The primary key itself will become the foreign key in the same table

Unary M:N

Example: Employee Guarantor

There will be two resulting tables. One to represent the entity and another to represent the many side

Binary 1:1

Example: Employee head of Department

The primary key of the partial participant will become the foreign key of the total participant. The primary key of either of the participants can become a foreign key in the other if the participation types are uniform (both total or both partial)

Binary 1:M

Example: Teacher teaches Subject

The primary key of the relation on the “1” side of the relationship becomes a foreign key in the relation on the “M” side

Binary M:N

Example: Books borrowed by Employee

A new table is created to represent the relationship

Contains two foreign keys – one from each of the participants in the relationship

The primary key of the new table is the combination of the two foreign keys

Ternary Relationship

Example: Doctor prescription contains Medicine given to Patients

The new table contains three foreign keys – one from each of the participating entities

The primary key of the new table is the combination of all three foreign keys

Test your Understanding

Map the E/R diagram of the educational institution in Session 4 to corresponding tables.

Session 6: Normalization

Learning Objectives

After completing this chapter, you will be able to:

- Define Normalisation
- Explain the role of Normalization in database design
- Explain the steps in Normalization: 1NF, 2NF, 3NF, BCNF
- Perform Normalisation

Introduction

We have considered the primary features of the relational database model. We have noted that relations that form the database must satisfy some properties, for example, relations have no duplicate tuples, tuples have no ordering associated with them, and each element in the relation is atomic. Relations that satisfy these basic requirements may still have some undesirable properties, for example, data redundancy and update anomalies. We illustrate these properties and study how relations may be transformed or decomposed (or *normalized*) to eliminate them. Most such undesirable properties do not arise if the database modeling has been carried out very carefully using some technique like the Entity-Relationship Model

The central concept is the notion of functional dependency which depends on understanding the semantics of the data and which deals with what information in a relation is dependent on what other information in the relation. We will define the concept of functional dependency and discuss how to reason with the dependencies. We will then show how to use the dependencies information to decompose relations whenever necessary to obtain relations that have the desirable properties that we want without loosing any of the information in the original relations.

Description of Normalization

Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating two factors: ***redundancy and inconsistent dependency.***

Redundant data wastes disk space and creates maintenance problems. If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. A change in the customer is much easier to implement if that data is stored only in the Customers table and nowhere else in the database.

What is an "inconsistent dependency"? While it is intuitive for a user to look in the Customers table for the address of a particular customer, it may not make sense to look there for the salary of the employee who calls on that customer. The employee's salary is related to, or dependent on, the employee and thus should be moved to the Employees table. Inconsistent dependencies can make data difficult to access; the path to find the data may be missing or broken.

There are a few rules for database normalization. Each rule is called a "normal form." If the first rule is observed, the database is said to be in "first normal form." If the first three rules are observed, the database is considered to be in "third normal form." Although other levels of normalization are possible, third normal form is considered the highest level necessary for most applications.

For example, in an Employee Recruitment table, a candidate's university name and address may be included. But you need a complete list of universities for group mailings. If university information is stored in the Candidates table, there is no way to list universities with no current candidates. Create a separate Universities table and link it to the Candidates table with a university code key.

Let us consider the following relation *student*. illustrated in Table 6.1

Table 6.1

Sno	sname	address	Cno	cname	instructor	office
85001	Smith	1, Main	CP302	Database	Gupta	102
85001	Smith	1, Main	CP303	Communication	Wilson	102
85001	Smith	1, Main	CP304	Software Engg.	Williams	1024
85005	Jones	12, 7th	CP302	Database	Gupta	102

The above table satisfies the properties of a relation and is said to be in *first normal form* (or *1NF*). Conceptually it is convenient to have all the information in one relation since it is then likely to be easier to query the database. But the above relation has the following undesirable features:

1. *Repetition of information* --- A lot of information is being repeated. Student name, address, course name, instructor name and office number are being repeated often. Every time we wish to insert a student enrolment, say, in CP302 we must insert the name of the course CP302 as well as the name and office number of its instructor. Also every time we insert a new enrolment for, say Smith, we must repeat his name and address. Repetition of information results in wastage of storage as well as other problems.
2. *Update Anomalies* --- Redundant information not only wastes storage but makes updates more difficult since, for example, changing the name of the instructor of CP302 would require that all tuples containing CP302 enrolment information be updated. If for some reason, all tuples are not updated, we might have a database that gives two names of instructor for subject CP302. This difficulty is called the *update anomaly*.
3. *Insertion Anomalies -- Inability to represent certain information* --- Let the primary key of the above relation be *(sno, cno)*. Any new tuple to be inserted in the relation must have a value for the primary key since existential integrity requires that a key may not be totally or partially NULL. However, if one wanted to insert the number and name of a new course in the database, it would not be possible until a student enrolls in the course and we are able to insert values of *sno* and *cno*. Similarly information about a new student cannot be inserted in the database until the student enrolls in a subject. These difficulties are called *insertion anomalies*.
4. Deletion Anomalies -- Loss of Useful Information --- In some instances, useful information may be lost when a tuple is deleted. For example, if we delete the tuple corresponding to student 85001 doing CP304, we will lose relevant information about course CP304 (viz. course name, instructor, office number) if the student 85001 was the only student enrolled in that course. Similarly deletion of course CP302 from the database may remove all information about the student named Jones. This is called *deletion anomalies*.

The above problems arise primarily because the relation student has information about students as well as subjects. One solution to deal with the problems is to decompose the relation into two or more smaller relations.

Decomposition may provide further benefits, for example, in a distributed database different relations may be stored at different sites if necessary. Of course, decomposition does increase the cost of query processing since the decomposed relations will need to be joined, sometime frequently.

The above relation may be easily decomposed into three relations to remove most of the above undesirable properties:

S (sno, sname, address)

C (cno, cname, instructor, office)

SC (sno, cno)

Such decomposition is called *normalization* and is essential if we wish to overcome undesirable anomalies. As noted earlier, normalization often has an adverse effect on performance. Data which could have been retrieved from one relation before normalization may require several relations to be joined after normalization. Normalization does however lead to more efficient updates since an update that may have required several tuples to be updated before normalization could well need only one tuple to be updated after normalization.

Although in the above case we are able to look at the original relation and propose a suitable decomposition that eliminates the anomalies that we have discussed, in general this approach is not possible. A relation may have one hundred or more attributes and it is then almost impossible for a person to conceptualize all the information and suggest a suitable decomposition to overcome the problems. We therefore need an algorithmic approach to finding if there are problems in a proposed database design and how to eliminate them if they exist.

There are several stages of the normalization process. These are called the *first normal form (1NF)*, the *second normal form (2NF)*, the *third normal form (3NF)*, *Boyce-Codd normal form (BCNF)*, the *fourth normal form (4NF)* and the *fifth normal form (5NF)*. For all practical purposes, 3NF or the BCNF are quite adequate since they remove the anomalies discussed above for most common situations. It should be clearly understood that there is no obligation to normalize relations to the highest possible level. Performance should be taken into account and this may result in a decision not to normalize, say, beyond second normal form.

Intuitively, the second and third normal forms are designed to result in relations such that each relation contains information about only one thing (either an entity or a relationship). That is, non-key attributes in each relation must provide a fact about the entity or relationship that is being identified by the key. Again, a sound E-R model of the database would ensure that all relations either provide facts about an entity or about a relationship resulting in the relations that are obtained being in 3NF.

It should be noted that decomposition of relations has to be always based on principles that ensure that the original relation may be reconstructed from the decomposed relations if and when necessary. If we are able to reduce redundancy and not loose any information, it implies that all that redundant information can be derived given the other information in the database. Therefore information that has been removed must be related or dependent on other information that still exists in the database. That is why the concept of redundancy is important. Careless decomposition of a relation can result in loss of information. We will discuss this in detail later.

Role of Normalization

In the E-R model, a conceptual schema using an entity- relationship diagram is built and then mapped to a set of relations. This technique ensures that each entity has information about only one thing and once the conceptual schema is mapped into a set of relations, each relation would have information about only one thing. The relations thus obtained would normally not suffer from any of the anomalies. This bottom-up approach is likely to lead to a relation that is likely to suffer from all the problemsn. For example, the relation is highly likely to have redundant information and update, deletion and insertion anomalies. Normalization of such large relation will then be essential to avoid (or at least minimize) these problems.

Now to define the normal forms more formally, we first need to define the concept of functional dependence.

Functional Dependency

Consider a relation *R* that has two attributes *A* and *B*. The attribute *B* of the relation is *functionally dependent* on the attribute *A* if and only if for each value of *A* no more than one value of *B* is associated. In other words, the value of attribute *A* uniquely determines the value of *B* and if there were several tuples that had the same value of *A* then all these tuples will have an identical value of attribute *B*.

If *B* is functionally dependent on *A* (or *A* functionally determines *B*).

For example, in the student database that we have discussed earlier, we have the following functional dependencies:

sno -> sname

sno -> address

cno -> cname

cno -> instructor

instructor -> office

These functional dependencies imply that there can be only one name for each *sno*, only one address for each student and only one subject name for each *cno*. It is of course possible that several students may have the same name and several students may live at the same address. If we consider ***cno -> instructor***, the dependency implies that no subject can have more than one instructor (perhaps this is not a very realistic assumption). Functional dependencies therefore place constraints on what information the database may store. In the above example, one may be wondering if the following FDs hold

sname -> sno

cname -> cno

First Normal Form (1NF)

A table satisfying the properties of a relation is said to be in first normal form. As discussed in an earlier chapter, a relation cannot have multivalued or composite attributes. This is what the 1NF requires.

A relation is in 1NF if and only if all underlying domains contain atomic values only.

The first normal form deals only with the basic structure of the relation and does not resolve the problems of redundant information or the anomalies discussed earlier. All relations discussed in these notes are in 1NF.

For example consider the following example relation:

student (sno, sname, dob)

Add some other attributes so it has anomalies and is not in 2NF

The attribute *dob* is the date of birth and the primary key of the relation is *sno* with the functional dependencies ***sno -> sname*** and ***sno -> dob***. The relation is in 1NF as long as *dob* is considered an atomic value and not consisting of three components *(day, month, year)*. The above relation of course suffers from all the anomalies that we have discussed earlier and needs to be normalized. (add example with date of birth)

Second Normal Form (2NF)

The second normal form attempts to deal with the problems that are identified with the relation above that is in 1NF. The aim of second normal form is to ensure that all information in one relation is only about one thing.

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.

To understand the above definition of 2NF we need to define the concept of *key attributes*. Each attribute of a relation that participates in at least one candidate key of is a *key* attribute of the relation. All other attributes are called *non-key*.

The concept of 2NF requires that all attributes that are not part of a candidate key be *fully* dependent on each candidate key. If we consider the relation

student (sno, sname, cno, cname)

and the functional dependencies

sno -> cname

cno -> cname

and assume that *(sno, cno)* is the only candidate key (and therefore the primary key), the relation is not in 2NF since *sname* and *cname* are not fully dependent on the key. The above relation suffers from the same anomalies and repetition of information as discussed above since *sname* and *cname* will be repeated. To resolve these difficulties we could remove those attributes from the relation that are not fully dependent on the candidate keys of the relations. Therefore we decompose the relation into the following projections of the original relation:

S1 (sno, sname)
S2 (cno, cname)
SC (sno, cno)

Use an example that leaves one relation in 2NF but not in 3NF.

We may recover the original relation by taking the natural join of the three relations.

If however we assume that *sname* and *cname* are unique and therefore we have the following candidate keys
(sno, cno)
(sno, cname)
(sname, cno)
(sname, cname)
The above relation is now in 2NF since the relation has no non-key attributes. The relation still has the same problems as before but it then does satisfy the requirements of 2NF. Higher level normalization is needed to resolve such problems with relations that are in 2NF and further normalization will result in decomposition of such relations.

Third Normal Form (3NF)

Although transforming a relation that is not in 2NF into a number of relations that are in 2NF removes many of the anomalies that appear in the relation that was not in 2NF, not all anomalies are removed and further normalization is sometime needed to ensure further removal of anomalies. These anomalies arise because a 2NF relation may have attributes that are not directly related to the thing that is being described by the candidate keys of the relation. Let us first define the 3NF.

A relation R is in third normal form if it is in 2NF and every non-key attribute of R is non-transitively dependent on each candidate key of R.

To understand the third normal form, we need to define *transitive dependence* which is based on one of Armstrong's axioms. Let *A*, *B* and *C* be three attributes of a relation *R* such that ***A -> B*** and ***B -> C***. From these FDs, we may derive ***A -> C***. As noted earlier, this dependence ***A -> C*** is *transitive*.

The 3NF differs from the 2NF in that all non-key attributes in 3NF are required to be *directly* dependent on each candidate key of the relation. The 3NF therefore insists, in the words of Kent (1983) that all facts in the relation are about the key (or the thing that the key identifies), the whole key and nothing but the key. If some attributes are dependent on the keys transitively then that is an indication that those attributes provide information not about the key but about a kno-key attribute. So the information is not directly about the key, although it obviously is related to the key.
Consider the following relation

subject (cno, cname, instructor, office)

Assume that *cname* is not unique and therefore *cno* is the only candidate key. The following functional dependencies exist

sno -> cname
cno -> instructor
instructor -> office

We can derive ***cno -> office*** from the above functional dependencies and therefore the above relation is in 2NF. The relation is however not in 3NF since office is not directly dependent on *cno*. This transitive dependence is an indication that the relation has information about more than one thing (viz. course and instructor) and should therefore be decomposed. The primary difficulty with the above relation is that an instructor might be responsible for several subjects and therefore his office address may need to be repeated many times. This leads to all the problems that we identified at the beginning of this chapter. To overcome these difficulties we need to decompose the above relation in the following two relations:

s (cno, cname, instructor)
ins (instructor, office)
s is now in 3NF and so is *ins*.

An alternate decomposition of the relation *subject* is possible:

s(cno, cname)
inst(instructor, office)
si(cno, instructor)

The decomposition into three relations is not necessary since the original relation is based on the assumption of one instructor for each course.

The 3NF is usually quite adequate for most relational database designs. There are however some situations, for example the relation *student(sno, sname, cno, cname)* discussed in 2NF above, where 3NF may not eliminate all the redundancies and inconsistencies. The problem with the relation *student(sno, sname, cno, cname)* is because of the redundant information in the candidate keys. These are resolved by further normalization using the BCNF.

Boyce-Codd Normal Form (BCNF)

The relation *student(sno, sname, cno, cname)* has all attributes participating in candidate keys since all the attributes are assumed to be unique. We therefore had the following candidate keys:

(sno, cno)

(sno, cname)

(sname, cno)

(sname, cname)

Since the relation has no non-key attributes, the relation is in 2NF and also in 3NF, in spite of the relation suffering the problems that we discussed at the beginning of this chapter.

The difficulty in this relation is being caused by dependence within the candidate keys. The second and third normal forms assume that all attributes not part of the candidate keys depend on the candidate keys but does not deal with dependencies *within* the keys. BCNF deals with such dependencies.

*A relation R is said to be in BCNF if whenever **X -> A** holds in R, and A is not in X, then X is a candidate key for R.*

It should be noted that most relations that are in 3NF are also in BCNF. Infrequently, a 3NF relation is not in BCNF and this happens only if

a. the candidate keys in the relation are composite keys (that is, they are not single attributes),
b. there is more than one candidate key in the relation, and
c. the keys are not disjoint, that is, some attributes in the keys are common.

The BCNF differs from the 3NF only when there are more than one candidate keys and the keys are composite and overlapping. Consider for example, the relationship

enrol (sno, sname, cno, cname, date-enrolled)

Let us assume that the relation has the following candidate keys:

(sno, cno)

(sno, cname)

(sname, cno)

(sname, cname)

(we have assumed sname and cname are unique identifiers). The relation is in 3NF but not in BCNF because there are dependencies

sno -> sname

cno -> cname

where attributes that are part of a candidate key are dependent on part of another candidate key. Such dependencies indicate that although the relation is about some entity or association that is identified by the candidate keys e.g. *(sno, cno)*, there are attributes that are not about the whole thing that the keys identify. For example, the above relation is about an association (enrolment) between students and subjects and therefore the relation needs to include only one identifier to identify students and one identifier to identify subjects. When there are two identifiers about students *(sno, sname)* and two keys about subjects *(cno, cname)* means that some information about students and subjects that is not needed is being provided. This provision of information will result in repetition of information and the anomalies. If we wish to include further information about students and courses in the database, it should not be done by putting the information in the present relation but by creating new relations that represent information about entities *student* and *subject*.

These difficulties may be overcome by decomposing the above relation in the following three relations:

(sno, sname)

(cno, cname)

(sno, cno, date-of-enrolment)

We now have a relation that only has information about students, another only about subjects and the third only about enrolments. All the anomalies and repetition of information have been removed.

Desirable properties of decomposition are:

1. Attribute preservation
2. Lossless-join decomposition
3. Dependency preservation
4. Lack of redundancy

Test your Understanding

1. A college keeps details about a student and the various modules the student has studied. These details comprise:
   - registration number
   - name
   - address
   - tutor number
   - tutor name
   - diploma code
   - diploma name

   and repeating fields for module code, module name and result.

   Reduce this data to third normal form.

2. Classify the following relations as unnormalised, INF, 2NF or 3NF. State any assumptions you make. If the relation is not in 3NF, normalise to 3NF.
   a. EMPLOYEE (EMP# , EMPNAME, JOBCODE)
   b. EMPLOYEE (EMP# , EMPNAME, (JOBCODE, YEARS))
   c. EMPLOYEE (EMP# , EMPNAME, JOBCODE, JOB DESCRIPTION)
   d. EMPLOYEE (EMP# , EMPNAME, PROJECT , HRS WORK)

Session 7: Structured Query Language – Basic Operations

Learning Objectives

After completing this chapter, you will be able to:

- Write SQL – Data Definition Language
- Write SQL – Data Manipulation Language
- Perform data retrieval using SQL Select statements

Introduction to SQL

Structured Query Language - (SQL) is the most widely used commercial relational database language. The SQL has several parts:

- DML – The Data Manipulation Language (DML)
- DDL – The Data Definition Language (DDL)
- Embedded and dynamic SQL
- Security
- Transaction management
- Client-server execution and remote database access

Objectives of SQL

Ideally, database language should allow user to:

- Create the database and relation structures;
- Perform insertion, modification, deletion of data from relations;
- Perform simple and complex queries.

It must perform these tasks with minimal user effort and command structure/syntax must be easy to learn.

SQL is relatively easy to learn:

- It is non-procedural - you specify *what* information you require, rather than *how* to get it;
- It is essentially free-format.

History of SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' (SEQUEL).
- Still pronounced 'see-quel', though official pronunciation is 'S-Q-L'.
- In late 70s, ORACLE appeared and was probably first commercial RDBMS based on SQL.
- In 1987, ANSI and ISO published an initial standard for SQL.

- In 1989, ISO published an addendum that defined an 'Integrity Enhancement Feature'.
- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.
- In 1999, SQL: 1999 was released with support for object-oriented data management.
- In late 2003, SQL: 2003 was released.

Writing SQL Commands

- SQL statement consists of *reserved words* and *user-defined words*.
  o. Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
  p. User-defined words are made up by user and represent names of various database objects such as relations, columns, views.
- Most components of an SQL statement are *case insensitive*, except for literal character data.
- More readable with indentation and lineation:
  q. Each clause should begin on a new line.
  r. Start of a clause should line up with start of other clauses.
  s. If clause has several parts, should each appear on a separate line and be indented under start of clause.
- Use extended form of BNF notation:
  t. Upper-case letters represent reserved words.
  u. Lower-case letters represent user-defined words.
  v. Indicates a *choice* among alternatives.
  w. Curly braces indicate a *required element*.
  x. Square brackets indicate an *optional element*.
  y. Indicates *optional repetition* (0 or more).
- Literals are constants used in SQL statements.
- All non-numeric literals must be enclosed in single quotes (e.g. 'London').
- All numeric literals must not be enclosed in quotes (e.g. 650.00).

SQL – Data Definition

DDL is used to create and modify the database objects like tables, views, indexes and sequences.

- Data types supported by SQL standard.

Table 7.1: ISO SQL Data types

Data type	Declarations			
boolean	BOOLEAN			
character	CHAR	VARCHAR		
bit	BIT	BIT VARYING		
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION	
datetime	DATE	TIME	TIMESTAMP	
interval	INTERVAL			
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT	

Integrity Enhancement Feature

- There are four types of integrity constraints:
  z. Required data
  aa. Domain constraints
  bb. Entity integrity
  cc. Referential integrity

Required Data

position VARCHAR(10) NOT NULL

Domain Constraints

(a) CHECK

Gender CHAR NOT NULL
CHECK (Gender IN ('M', 'F'))

- *searchCondition* can involve a table lookup:

  CREATE DOMAIN BranchNo AS CHAR (4)
  CHECK (VALUE IN (SELECT branchNo FROM Branch));

- Domains can be removed using DROP DOMAIN:

  DROP DOMAIN DomainName
  [RESTRICT | CASCADE]

Entity Integrity

- Primary key of a table must contain a unique, non-null value for each row.
- ISO standard supports FOREIGN KEY clause in CREATE and ALTER TABLE statements:

  PRIMARY KEY (staffNo)
  PRIMARY KEY (clientNo, propertyNo)

- Can only have one PRIMARY KEY clause per table. Can still ensure uniqueness for alternate keys using UNIQUE:

UNIQUE(telNo)

Referential Integrity

- FK is column or set of columns that links each row in child table containing foreign FK to row of parent table containing matching PK.
- Referential integrity means that, if FK contains a value, that value must refer to existing row in parent table.
- ISO standard supports definition of FKs with FOREIGN KEY clause in CREATE and ALTER TABLE:

  FOREIGN KEY(branchNo) REFERENCES Branch
- Any INSERT/UPDATE attempting to create FK value in child table without matching CK value in parent is rejected.
- Action taken attempting to update/delete a CK value in parent table with matching rows in child is dependent on referential action specified using ON UPDATE and ON DELETE subclauses:

  dd. CASCADE - SET NULL

  ee. SET DEFAULT - NO ACTION

CASCADE: Delete row from parent and delete matching rows in child, and so on in cascading manner.

SET NULL: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns are NOT NULL.

SET DEFAULT: Delete row from parent and set each component of FK in child to specified default. This is valid only if DEFAULT specified for FK columns.

NO ACTION: Reject delete from parent. This is the default setting.

FOREIGN KEY (staffNo) REFERENCES Staff ON DELETE SET NULL

FOREIGN KEY (ownerNo) REFERENCES Owner ON UPDATE CASCADE

CREATE TABLE

CREATE TABLE

- Creates a table with one or more columns of the specified *dataType*.
- With NOT NULL, system rejects any attempt to insert a null in the column.
- Can specify a DEFAULT value for the column.
- Primary keys should always be specified as NOT NULL.
- FOREIGN KEY clause specifies FK along with the referential action.

Syntax for CREATE TABLE

CREATE TABLE TableName
{(colName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] […,]}
{[FOREIGN KEY (listOfFKColumns)
REFERENCES ParentTableName [(listOfCKColumns)],
[ON UPDATE referentialAction]
[ON DELETE referentialAction]] [,…]}
{[CHECK (searchCondition)] [,…] })

Example - CREATE TABLE

```
CREATE TABLE PropertyForRent (
        propertyNo      PNumber NOT NULL, ….
        rooms PRooms NOT NULL       DEFAULT 4,
        rent     PRent NOT NULL, DEFAULT 600,
        ownerNo OwnerNumber NOT NULL,
        staffNo StaffNumber                                    Constraint
StaffNotHandlingTooMuch ….
        branchNo BranchNumber NOT NULL,
        PRIMARY KEY (propertyNo),
        FOREIGN KEY (staffNo) REFERENCES Staff
ON DELETE SET NULL ON UPDATE CASCADE ….);
```

ALTER TABLE

ALTER TABLE

- Add a new column to a table.
- Drop a column from a table.
- Add a new table constraint.
- Drop a table constraint.
- Set a default for a column.
- Drop a default for a column.

ALTER TABLE

Change Staff table by removing default of ‘Assistant’ for position column and setting default for sex column to female (‘F’).

```
        ALTER TABLE Staff
                Modify position DROP DEFAULT;
            ALTER TABLE Staff
                MODIFY sex SET DEFAULT ‘F’;
```

ALTER TABLE

Remove constraint from PropertyForRent that staff are not allowed to handle more than 100 properties at a time. Add new column to Client table.

ALTER TABLE PropertyForRent DROP CONSTRAINT StaffNotHandlingTooMuch;
ALTER TABLE Client ADD prefNoRooms PRooms;

DROP TABLE

DROP TABLE TableName [RESTRICT | CASCADE]

Example: DROP TABLE PropertyForRent;

- Removes named table and all rows within it.
- With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.
- With CASCADE, SQL drops all dependent objects (and objects dependent on these objects).

SQL – Data Manipulation

Data Manipulation

- INSERT
- UPDATE
- DELETE

INSERT

Syntax for Insert:

INSERT INTO TableName [(columnList)]
VALUES (dataValueList)

- *columnList* is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.
- *dataValueList* must match *columnList* as follows:
  ff. number of items in each list must be same;
  gg. must be direct correspondence in position of items in two lists;
  hh. data type of each item in *dataValueList* must be compatible with data type of corresponding column.

Example:

ii. Insert a new row into Staff table supplying data for all columns.

INSERT INTO Staff
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M', Date'1957-05-25', 8300, 'B003');
Insert a new row into Staff table supplying data for all mandatory columns.

iii. Insert a new row into Staff table supplying data for all mandatory columns

INSERT INTO Staff (staffNo, fName, lName, position, salary, branchNo)
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', 8100, 'B003');
Or
INSERT INTO Staff
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', NULL, NULL, 8100, 'B003');

INSERT … SELECT

Second form of INSERT allows multiple rows to be copied from one or more tables to another:

INSERT INTO TableName [(columnList)]
SELECT ...

Assume there is a table StaffPropCount that contains names of staff and number of properties they manage:

StaffPropCount(staffNo, fName, lName, propCnt)
Populate StaffPropCount using Staff and PropertyForRent tables.

UPDATE

Syntax:

UPDATE TableName
SET columnName1 = dataValue1
[, columnName2 = dataValue2...]
[WHERE searchCondition]

- *TableName* can be name of a base table or an updatable view.
- SET clause specifies names of one or more columns that are to be updated.
- WHERE clause is optional:
  a. if omitted, named columns are updated for all rows in table;
  b. if specified, only those rows that satisfy *searchCondition* are updated.
- New *dataValue(s)* must be compatible with data type for corresponding column.

UPDATE All Rows
Give all staff a 3% pay increase.

UPDATE Staff
SET salary = salary*1.03;

Give all Managers a 5% pay increase.

UPDATE Staff
SET salary = salary*1.05
WHERE position = 'Manager';

UPDATE Multiple Columns
Promote David Ford (staffNo='SG14') to Manager and change his salary to £18,000.

UPDATE Staff
SET position = 'Manager', salary = 18000
WHERE staffNo = 'SG14';

DELETE

Syntax:
DELETE FROM TableName
[WHERE searchCondition]

- *TableName* can be name of a base table or an updatable view.
- *searchCondition* is optional; if omitted, all rows are deleted from table. This does not delete table. If *search_condition* is specified, only those rows that satisfy condition are deleted.

DELETE Specific Rows
Delete all viewings that relate to property PG4.

DELETE FROM Viewing
WHERE propertyNo = 'PG4';

SQL – SELECT Statements

Select Statement is used to retrieve rows from the tables in a database.

Syntax:

SELECT [DISTINCT | ALL]
{* | [columnExpression [AS newName]] [,...] }
FROM TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList]
[HAVING condition]
[ORDER BY columnList]
FROM Specifies table(s) to be used.
WHERE Filters rows.
GROUP BY Forms groups of rows with same column value.
HAVING Filters groups subject to some condition.
SELECT Specifies which columns are to appear in output.
ORDER BY Specifies the order of the output.

- Order of the clauses cannot be changed.
- Only SELECT and FROM are mandatory.

Examples of Select Statements

Example 7.1 All Columns, All Rows

List full details of all staff:

SELECT staffNo, fName, lName, address, position, sex, DOB, salary, branchNo
FROM Staff;

Can use * as an abbreviation for 'all columns'

SELECT * FROM Staff;

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

Example 7.2 Specific Columns, All Rows

Produce a list of salaries for all staff, showing only staff number, first and last names, and salary.

SELECT staffNo, fName, lName, salary FROM Staff;

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

Example 7.3 Use of DISTINCT

List the property numbers of all properties that have been viewed.

SELECT propertyNo FROM Viewing;

propertyNo
PA14
PG4
PG4
PA14
PG36

Use DISTINCT to eliminate duplicates:

SELECT DISTINCT propertyNo FROM Viewing;

propertyNo
PA14
PG4
PG36

Example 7.4 Calculated Fields

Produce list of monthly salaries for all staff, showing staff number, first/last name, and salary.

SELECT staffNo, fName, lName, salary/12 FROM Staff;

staffNo	fName	lName	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

To name column, use AS clause:

SELECT staffNo, fName, lName, salary/12
AS monthlySalary FROM Staff;

Example 7.5 Comparison Search Condition

List all staff with a salary greater than 10,000.

SELECT staffNo, fName, lName, position, salary FROM Staff
WHERE salary > 10000;

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Example 7.6 Compound Comparison Search Condition

List addresses of all branch offices in London or Glasgow.

SELECT * FROM Branch WHERE city = 'London' OR city = 'Glasgow';

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

Example 7.7 Range Search Condition

List all staff with a salary between 20,000 and 30,000.

SELECT staffNo, fName, lName, position, salary FROM Staff
WHERE salary BETWEEN 20000 AND 30000;

BETWEEN test includes the endpoints of range.
Also a negated version NOT BETWEEN.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

SELECT staffNo, fName, lName, position, salary FROM Staff
WHERE salary>=20000 AND salary <= 30000;

Example 7.8 Set Membership

List all managers and supervisors.

SELECT staffNo, fName, lName, position FROM Staff
WHERE position IN ('Manager', 'Supervisor');

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

There is a negated version (NOT IN).

SELECT staffNo, fName, lName, position FROM Staff
WHERE position='Manager' OR position='Supervisor';

Example 7.9 Pattern Matching

Find all owners with the string 'Glasgow' in their address.

SELECT ownerNo, fName, lName, address, telNo FROM PrivateOwner
WHERE address LIKE '%Glasgow%';

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

SQL has two special pattern matching symbols:
%: sequence of zero or more characters;
_ (underscore): any single character.
LIKE '%Glasgow%' means a sequence of characters of any length containing '*Glasgow*'.

Example 7.10 NULL Search Condition

List details of all viewings on property PG4 where a comment has not been supplied.

- There are 2 viewings for property PG4, one with and one without a comment.
- Have to test for null explicitly using special keyword IS NULL:

SELECT clientNo, viewDate FROM Viewing WHERE propertyNo = 'PG4' AND comment IS NULL;

clientNo	viewDate
CR56	26-May-04

Negated version (IS NOT NULL) can test for non-null values.

Example 7.11 Single Column Ordering

List salaries for all staff, arranged in descending order of salary.

SELECT staffNo, fName, lName, salary FROM Staff
ORDER BY salary DESC;

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

Example 7.12 Multiple Column Ordering

Produce abbreviated list of properties in order of property type.

```
SELECT propertyNo, type, rooms, rent   FROM PropertyForRent
ORDER BY type;
```

propertyNo	type	rooms	rent
PL94	Flat	4	400
PG4	Flat	3	350
PG36	Flat	3	375
PG16	Flat	4	450
PA14	House	6	650
PG21	House	5	600

- Four flats in this list - as no minor sort key specified, system arranges these rows in any order it chooses.
- To arrange in order of rent, specify minor order:

```
SELECT propertyNo, type, rooms, rent FROM PropertyForRent
ORDER BY type, rent DESC;
```

propertyNo	type	rooms	rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600

SELECT Statement – Aggregate Functions

ISO standard defines five aggregate functions:

COUNT returns number of values in specified column.

SUM returns sum of values in specified column.

AVG returns average of values in specified column.

MIN returns smallest value in specified column.

MAX returns largest value in specified column.

- Each operates on a single column of a table and returns a single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG may be used on numeric fields only.
- Apart from COUNT(*), each function eliminates nulls first and operates only on remaining non-null values.
- COUNT(*) counts all rows of a table, regardless of whether nulls or duplicate values occur.
- Can use DISTINCT before column name to eliminate duplicates.
- DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.
- Aggregate functions can be used only in SELECT list and in HAVING clause.
- If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference a column out with an aggregate function. For example, the following is illegal:

```
SELECT staffNo, COUNT(salary) FROM Staff;
```

Example 7.13 Use of COUNT(*)

How many properties cost more than £350 per month to rent?

```
SELECT COUNT(*) AS myCount FROM PropertyForRent
WHERE rent > 350;
```

myCount
5

Example 7.14 Use of COUNT(DISTINCT)

How many different properties viewed in May ‘04?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
```

WHERE viewDate BETWEEN '1-May-04' AND '31-May-04';

myCount
2

Example 7.15 Use of COUNT and SUM
Find number of Managers and sum of their salaries.

SELECT COUNT(staffNo) AS myCount, SUM(salary) AS mySum
FROM Staff WHERE position = 'Manager';

myCount	mySum
2	54000.00

Example 7.16 Use of MIN, MAX, AVG
Find minimum, maximum, and average staff salary.

SELECT MIN(salary) AS myMin, MAX(salary) AS myMax,
AVG(salary) AS myAvg FROM Staff;

myMin	myMax	myAvg
9000.00	30000.00	17000.00

SELECT Statement - Grouping

- Use GROUP BY clause to get sub-totals.
- SELECT and GROUP BY closely integrated: each item in SELECT list must be *single-valued per group*, and SELECT clause may only contain:
  c. Column names
  d. Aggregate functions
  e. Constants
  f. Expression involving combinations of the above.
- All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.
- If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ISO considers two nulls to be equal for purposes of GROUP BY.

Example 7.17 Use of GROUP BY

Find number of staff in each branch and their total salaries.

```
SELECT branchNo, COUNT(staffNo) AS myCount, SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

Restricted Groupings – HAVING clause

- HAVING clause is designed for use with GROUP BY to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.

Example 7.18 Use of HAVING

For each branch with more than 1 member of staff, find number of staff in each branch and sum of their salaries.

```
SELECT branchNo, COUNT(staffNo) AS myCount, SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Subqueries

- Some SQL statements can have a SELECT embedded within them.
- A subselect can be used in WHERE and HAVING clauses of an outer SELECT, where it is called a *subquery* or *nested query*.
- Subselects may also appear in INSERT, UPDATE, and DELETE statements.

Example 7.19 Subquery with Equality

List staff who work in branch at '163 Main St'.

```
SELECT staffNo, fName, lName, position FROM Staff
WHERE branchNo = (SELECT branchNo FROM Branch
WHERE street = '163 Main St');
```

- Inner SELECT finds branch number for branch at '163 Main St' ('B003').
- Outer SELECT then retrieves details of all staff who work at this branch.
- Outer SELECT then becomes:

```
SELECT staffNo, fName, lName, positionFROM Staff
WHERE branchNo = 'B003';
```

staffNo	fName	lName	position
SG37	Ann	Beech	Assistant
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Example 7.20 Subquery with Aggregate

List all staff whose salary is greater than the average salary, and show by how much.

```
SELECT staffNo, fName, lName, position,
  salary – (SELECT AVG(salary) FROM Staff) As SalDiff
FROM Staff
WHERE salary > (SELECT AVG(salary)  FROM Staff);
```

- Cannot write 'WHERE salary > AVG(salary)'
- Instead, use subquery to find average salary (17000), and then use outer SELECT to find those staff with salary greater than this:

```
SELECT staffNo, fName, lName, position,  salary – 17000 As salDiff
FROM Staff
WHERE salary > 17000;
```

Subquery Rules

- ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).
- Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.
- By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an *alias*.
- When subquery is an operand in a comparison, subquery must appear on right-hand side.
- A subquery may not be used as an operand in an expression.

Test your Understanding

1. What is the keyword used to rename a column in a table in the select statement?
2. Can you increase the size of a column using ALTER table command?
3. Is it possible to use ORDER BY clause in a Subquery
4. Can we use an aggregate function in a WHERE clause?
5. Can we delete all the columns in a table

Answers

1. AS
2. YES
3. No
4. No
5. No

Session 10: Structured Query Language – Advanced Operations

Learning Objectives

After completing this chapter, you will be able to:

- Explain SQL Joins
- Describe other database Objects – Indexes, Sequences, and Views

SQL JOINS

Joining Tables

To appropriately join tables, the tables must be related and we apply a where clause which equates the primary key column of the table on the one side of the relationship with the parallel foreign key column of the many side table.

Joining Two Tables - SQL will compare every row of the 1st table with the first row of the 2nd table. Then it will compare all rows of the 1st with the second row of the second, and so on only rows where the condition is met are placed in the result table.

Procedure for generating results of a join is:

1. Form Cartesian product of the tables named in FROM clause.
2. If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition.
3. For each remaining row, determine value of each item in SELECT list to produce a single row in result table.
4. If DISTINCT has been specified, eliminate any duplicate rows from the result table.
5. If there is an ORDER BY clause, sort result table as required.
    - SQL provides special format of SELECT for Cartesian product:

```
SELECT          [DISTINCT | ALL]          {* | columnList}
FROM Table1 CROSS JOIN Table2
```

SQL Indexes

An Index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically. If there is no index on the column then a full table scan occurs. An index provides fast access to rows in a table. Indexes are logically and physically independent of the table they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes. When a table is dropped the corresponding indexes are also dropped.

When to create an index :

- If the column contains wide range of values
- If the column contains large number of null values
- If one or more columns are frequently used together in a WHERE clause or join condition

To create an index

CREATE INDEX index ON TABLE (column[,column]…);

SQL Sequences

A sequence is a user created database object that can be shared by multiple users to generate unique integers. A typical usage for sequences is to create primary key value which must be unique for each row.

CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n I NOMAXVALUE}]
[{MINVALUE n I NOMINVALUE}]
[{CYCLE I NOCYCLE}]
[{CACHE I NOCACHE}];

SQL Views

A View is a dynamic result of one or more relational operations operating on base relations to produce another relation.

- Virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.
- Contents of a view are defined as a query on one or more base relations.
- With view resolution, any operations on view are automatically translated into operations on relations from which it is derived.
- With view materialization, the view is stored as a temporary table, which is maintained as the underlying base tables are updated.

Syntax for creating view

CREATE VIEW ViewName [(newColumnName [,...])]
AS subselect
[WITH [CASCADED | LOCAL] CHECK OPTION]

- Can assign a name to each column in view.
- If list of column names is specified, it must have same number of items as number of columns produced by *subselect*.
- If omitted, each column takes name of corresponding column in *subselect*.
- List must be specified if there is any ambiguity in a column name.
- The *subselect* is known as the defining query.
- WITH CHECK OPTION ensures that if a row fails to satisfy WHERE clause of defining query, it is not added to underlying base table.
- Need SELECT privilege on all tables referenced in subselect and USAGE privilege on any domains used in referenced columns.

Grouped and Joined Views

Create view of staff who manage properties for rent, including branch number they work at, staff number, and number of properties they manage.

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
                FROM Staff s, PropertyForRent p
                WHERE s.staffNo = p.staffNo
                GROUP BY s.branchNo, s.staffNo;
```

DROP VIEW ViewName [RESTRICT | CASCADE]

- Causes definition of view to be deleted from database.
- For example:
  DROP VIEW Manager3Staff;
- With CASCADE, all related dependent objects are deleted; i.e. any views defined on view being dropped.
- With RESTRICT (default), if any other objects depend for their existence on continued existence of view being dropped, command is rejected.

Restrictions on Views

SQL imposes several restrictions on creation and use of views.

(a) If column in view is based on an aggregate function:

- Column may appear only in SELECT and ORDER BY clauses of queries that access view.
- Column may not be used in WHERE nor be an argument to an aggregate function in any query based on view.
- For example, following query would fail:

  SELECT COUNT(cnt)

  FROM StaffPropCnt;
- Similarly, following query would also fail:

  SELECT *

  FROM StaffPropCnt

  WHERE cnt > 2;

(b) Grouped view may never be joined with a base table or a view.

- For example, StaffPropCnt view is a grouped view, so any attempt to join this view with another table or view fails.

Test your Understanding

1. Is it possible to make changes in the base table by updating in the view? Ans. Yes
2. How many join conditions are needed for joining 3 tables? Ans. 2
3. Does the select statement retrieve rows directly from the tables? Ans. No

Session 13: File Organization and Database Tuning

Learning Objectives

After completing this chapter, you will be able to:

- Explain the File Organisation
- Explain the Performance Tuning of the Database

File Organization

Database is stored internally in the secondary storage devices like tapes and disks. Since disks are random access devices, we will see the file organization of database in the disks. Disks are divided into concentric circular tracks on each disk surface. Track capacities vary typically from 4 to 50 Kbytes. Because a track usually contains a large amount of information, it is divided into smaller blocks or sectors. The division of a track into sectors is hard-coded on the disk surface and cannot be changed.

A track is divided into blocks. The block size B is fixed for each system. Typical block sizes range from B=512 bytes to B=4096 bytes. Whole blocks are transferred between disk and main memory for processing.

Records

The records can be either Fixed or variable length records. When records contain fields which have values of a particular type (e.g., amount, date, time, age) they are called fixed length records while fields themselves may be variable length such that of an address field, are called variable length record.

Files of Records

A file is a sequence of records, where each record is a collection of data values (or data items).A file descriptor (or file header) includes information that describes the file, such as the field names and their data types, and the addresses of the file blocks on disk. Records are stored on disk blocks. The blocking factor for a file is the (average) number of file records stored in a disk block. A file can have fixed-length records or variable-length records. File records can be unspanned (no record can span two blocks) or spanned (a record can be stored in more than one block).The physical disk blocks that are allocated to hold the records of a file can be contiguous, linked, or indexed. In a file of fixed-length records, all records have the same format. Usually, unspanned blocking is used with such files. Files of variable-length records require additional information to be stored in each record, such as separator characters and field types. Usually spanned blocking is used with such files.

So for a database to store data, it needs computer system as well as operating system. The aspect of storing data and file management is taken care of by the Operating system. So the two main components of an OS which is needed for a DBMS are file manager and disk manager.

When a new row has to be inserted the file manager requests the disk manager who in turn allocates certain memory space to the file manager. The file manager interacts with the DBMS and the disk manager interacts with the low level and does the logical to the physical mapping of address.

DBMS/Host inter-communication

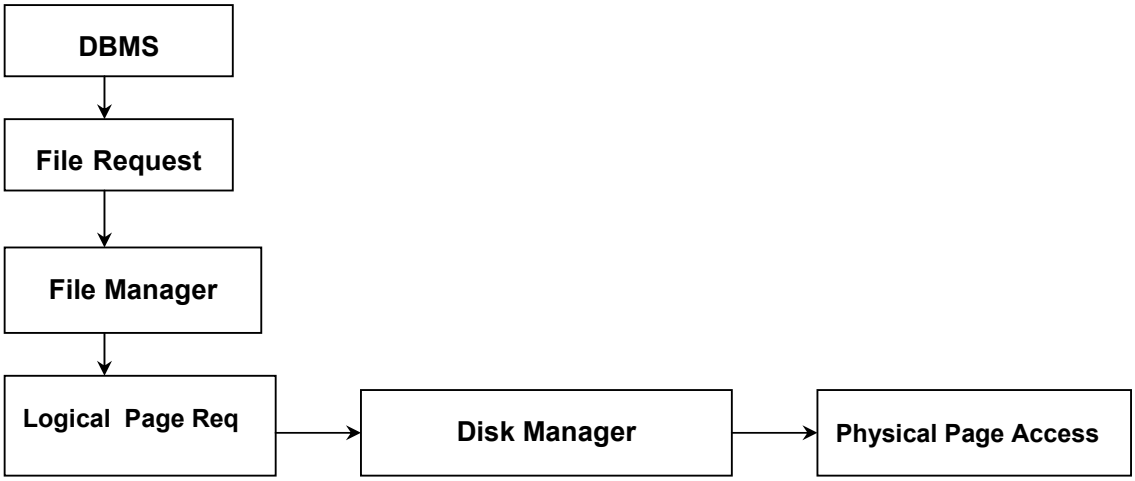


Fig. 13.1: DBMS/Host inter-communication

Database Tuning

Tuning refers to the process of continuing to revise/adjust the physical database design by monitoring resource utilization as well as internal DBMS processing to reveal bottlenecks such as contention for the same data or devices.

The goals of database tuning are -

- To make application run faster
- To lower the response time of queries/transactions
- To improve the overall throughput of transactions

The main issues to be considered in tuning are:

- How to avoid excessive lock contention?
- How to minimize overhead of logging and unnecessary dumping of data?
- How to optimize buffer size and scheduling of processes?
- How to allocate resources such as disks, RAM and processes for most efficient utilization?

So the different strategies are

- Creating Indexes
- Tuning the Database design
- Tuning Queries

Tuning using Indexes

The main reasons to tuning using indexes are

- Certain queries may take too long to run for lack of an index;
- Certain indexes may not get utilized at all;
- Certain indexes may be causing excessive overhead because the index is on an attribute that undergoes frequent changes

The various Options to tuning using indexes are

- Drop or/and build new indexes
- Change a non-clustered index to a clustered index (and vice versa)
- Rebuilding the index

Tuning the Database Design

Dynamically changed processing requirements need to be addressed by making changes to the conceptual schema if necessary and to reflect those changes into the logical schema and physical design.

The possible changes to the database design

- Existing tables may be joined (denormalized) because certain attributes from two or more tables are frequently needed together.
- For the given set of tables, there may be alternative design choices, all of which achieve 3NF or BCNF. One may be replaced by the other.
- A relation of the form R (K, A, B, C, D …) that is in BCNF can be stored into multiple tables that are also in BCNF by replicating the key K in each table.
- Attribute(s) from one table may be repeated in another even though this creates redundancy and potential anomalies.
- Apply horizontal partitioning as well as vertical partitioning if necessary.

Tuning Queries

Indications for tuning queries

- A query issues too many disk accesses
- The query plan shows that relevant indexes are not being used.

Typical instances for query tuning

- Many query optimizers do not use indexes in the presence of arithmetic expressions, numerical comparisons of attributes of different sizes and precision, NULL comparisons, and sub-string comparisons.
- Indexes are often not used for nested queries using IN;
- Some DISTINCTs may be redundant and can be avoided without changing the result.
- Unnecessary use of temporary result tables can be avoided by collapsing multiple queries into a single query unless the temporary relation is needed for some intermediate processing.
- In some situations involving using of correlated queries, temporaries are useful.

- If multiple options for join condition are possible, choose one that uses a clustering index and avoid those that contain string comparisons.
- The order of tables in the FROM clause may affect the join processing.
- Some query optimizers perform worse on nested queries compared to their equivalent un-nested counterparts.
- Many applications are based on views that define the data of interest to those applications. Sometimes these views become an overkill.

Test your Understanding

1. What is the role of the disk manager?

2. What is a block?

3. Is index good for database tuning?

4. How is performance measured?

Session 14: Emerging Trends

Learning Objectives

After completing this chapter, you will be able to:

- Identify the latest trends in the Database Arena

Distributed Database Management System

We have been concentrating on centralized database where data is stored centrally on a mainframe or minicomputer, accessed via dumb terminals. Though it serves the purpose of structured information, it fails when it comes to fast access to data and information i.e. it does not support query to generate on the spot information.

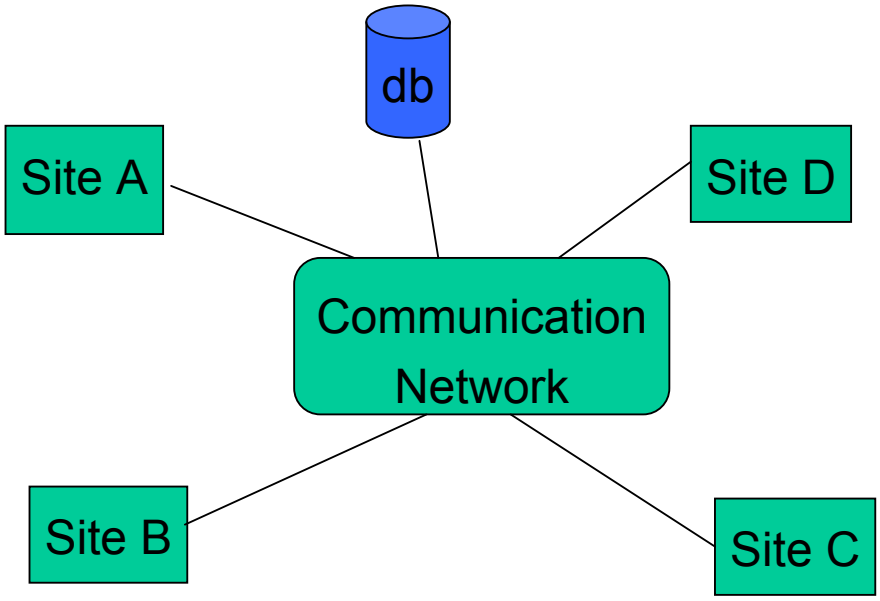


Fig 14.1: Centralized Databases

The main drawbacks of centralized databases are

- Performance degradation due to greater distance.
- High cost of mainframes, its maintenance and operations.
- Reliability problem
- Everybody is dependent on a central site, as a result if the server goes down the entire network stops functioning.

As a result of the above problem, the need to have distributed database management system (DDBMS) emerged.

In Distributed Database Environment data is stored at different i.e.. *more than one* location

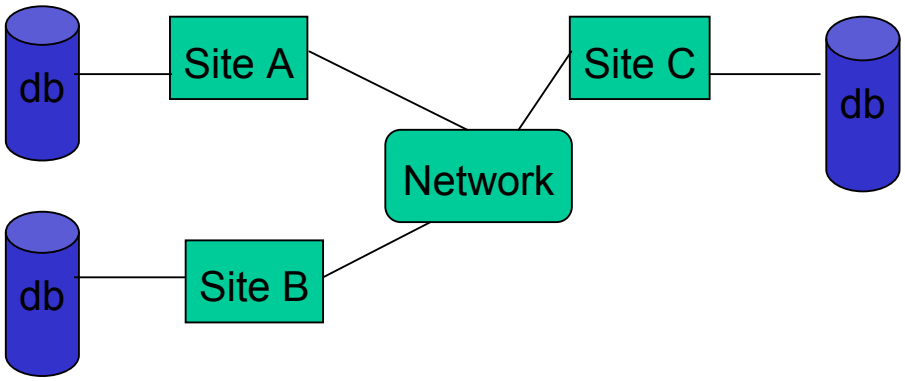


Fig 14.2: Distributed Database Environment

A fully DDBMS governs the storage and processing of logically related data over interconnected computer systems in which both data and processing function are distributed among several sites.

Advantages of DDBMS

- Faster access to data.
- Improved communication.
- Cost effective – as cost of dedicated communication lines and mainframes is reduced.
- Less danger of single point failure.
- Processor independence – available processor may process the user's request.

Disadvantages of DDBMS

- Complex in data management and control – distributed data across various locations and maintaining location transparency is difficult. Transaction management, concurrency control, security, backup, recovery, query optimization must all be addressed and resolved.
- Lack of standard protocol for communication.

Types of DDBMS

- Homogeneous DDBMS – In this type of DDBMS all the servers on which data is distributed run the same DBMS software.
- Heterogeneous DDBMS - In this type of DDBMS all the servers on which data is distributed run different DBMS software I.e. a multidatabase system. Such networks must support standards for gateway protocol. A gateway protocol is an API (Application Programming Interface) that exposes DBMS functionality external applications. Eg ODBC and JDBC. However gateway is an additional processing layer that can be expensive.

Data Distribution Strategies

Data Fragmentation

A single object can be broken into two or more segments or fragments. The details about fragments are maintained in the distributed data catalog (DDC).

Data Fragmentation may be

- Horizontal – subset of tuples
- Vertical – attribute subset.(Key repeated)
- Mixed

Data Replication

Storing the same data copies at multiple location or sites is called data replication.

Replication of data is subjected to mutual consistency rule which means they must be identical. This increases availability of data and faster query evaluation.

C J Date's 12 Rules of Dependency

- Local site Independence – Each site should be self sufficient and take care of ascepts like security, concurrency, backup, recovery, etc.
- Central site Independence – No site should be dependent on central or any other site.
- Failure Independence – should continue operations even if a node or server fails.
- Location Transparency – location of data need not be know to user.
- Fragmentation Transparency – users should see single logical database. They need not know the name of fragment.
- Replication Transparency
- Distributed query processing – Query may be executed at several different data processing sites.
- Distributed transaction processing
- Operating system Independence
- Network Independence
- Database Independence

Data Warehousing

Since 1970s, organizations gained competitive advantage through systems that automate business processes to offer more efficient and cost-effective services to the customer. This resulted in accumulation of growing amounts of data in operational databases. Organizations now focus on ways to use operational data to support decision-making, as a means of gaining competitive advantage. However, operational systems were never designed to support such business activities.

Businesses typically have numerous operational systems with overlapping and sometimes contradictory definitions. Organizations need to turn their archives of data into a source of knowledge, so that a single integrated / consolidated view of the organization's data is presented to the user.

A data warehouse was deemed the solution to meet the requirements of a system capable of supporting decision-making, receiving data from multiple operational data sources. Data warehouse can be thought of as a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision-making process (Inmon, 1993).

The warehouse is organized around the major subjects of the enterprise (e.g. customers, products, and sales) rather than the major application areas (e.g. customer invoicing, stock control, and product sales). This is reflected in the need to store decision-support data rather than application-oriented data. The data warehouse integrates corporate application-oriented data from different source systems, which often includes data that is inconsistent.

The integrated data source must be made consistent to present a unified view of the data to the users. Data in the warehouse is only accurate and valid at some point in time or over some time interval. Time-variance is also shown in the extended time that the data is held, the implicit or explicit association of time with all data, and the fact that the data represents a series of snapshots. Data in the warehouse is not updated in real-time but is refreshed from operational systems on a regular basis. New data is always added as a supplement to the database, rather than a replacement.

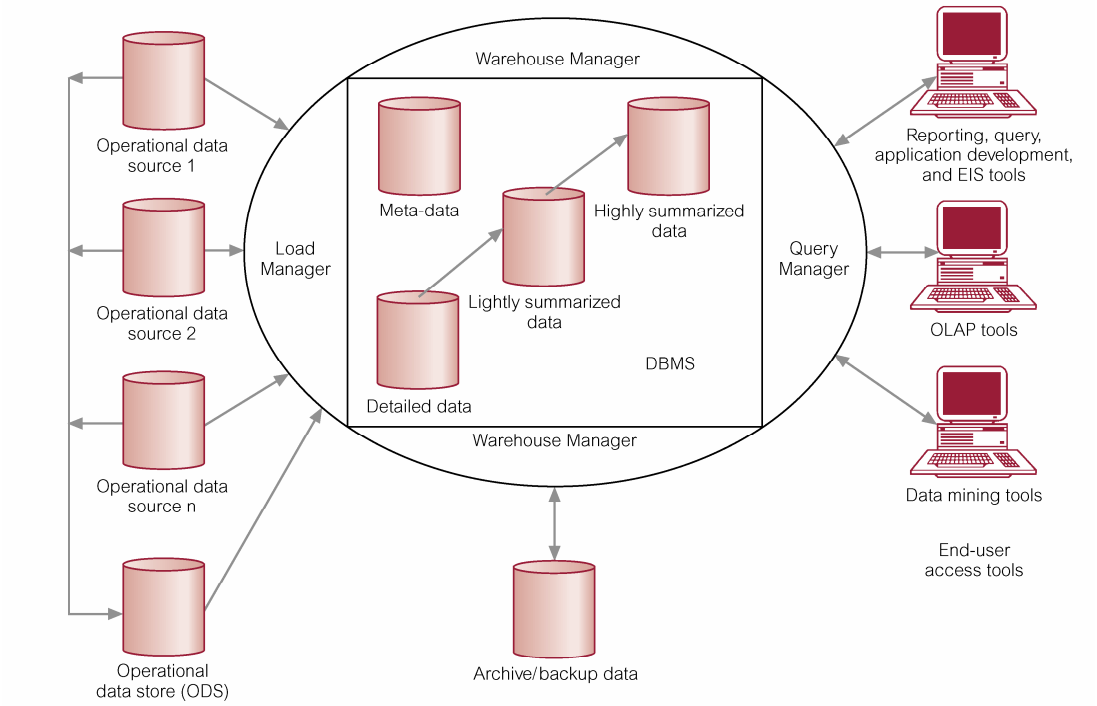


Fig 14.3 Data warehouse Architecture

Data mining Concepts

The process of extracting valid, previously unknown, comprehensible, and actionable information from large databases and using it to make crucial business decisions, (Simoudis,1996).It involves the analysis of data and the use of software techniques for finding hidden and unexpected patterns and relationships in sets of data. Data mining reveals information that is hidden and unexpected, as little value in finding patterns and relationships that are already intuitive. Patterns and relationships are identified by examining the underlying rules and features in the data. It tends to work from the data up and most accurate results normally require large volumes of data to deliver reliable conclusions. The entire process starts by developing an optimal representation of structure of sample data, during which time knowledge is acquired and extended to larger sets of data.

Data mining can provide huge paybacks for companies who have made a significant investment in data warehousing. It is relatively a new technology, however already used in a number of industries.

One Industry application is as follows

- Retail / Marketing
- Identifying buying patterns of customers
- Finding associations among customer demographic characteristics
- Predicting response to mailing campaigns
- Market basket analysis

The four main operations include:

- Predictive modeling
- Database segmentation
- Link analysis
- Deviation detection

There are recognized associations between the applications and the corresponding operations.
e.g. Direct marketing strategies use database segmentation.

Techniques are specific implementations of the data mining operations.
Each operation has its own strengths and weaknesses.

Criteria for selection of the proper data mining technique includes

- Suitability for certain input data types
- Transparency of the mining output
- Tolerance of missing variable values
- Level of accuracy possible
- Ability to handle large volumes of data

Goals of data mining:

- Quickly find association rules over extremely large data sets (For example: All Wal-Mart sales for a year).
- Allow user to tune support and confidence.
- Also other types of rules and patterns.

Test your Understanding

1. Give an application of DDBMS.

2. Give an application for Data Warehousing.

3. Give an application of Datamining.

Glossary

Attribute
An attribute is a named column of a relation.

Candidate key:
An attribute or group of attributes that identifies a unique row in a relation. One of the candidate keys is chosen to be the primary key.

Cardinality
The cardinality of a relation is the number of tuples it contains.

Child:
A row, record, or node on the “many” side of a one-to-many relationship.

Composite key
A key of a relation that consists of two or more columns.

Concurrency
A condition in which two or more transactions are processed against a database at the same time. In a single CPU system, the changes are interleaved; in a multi-CPU system, the transactions can be processed simultaneously, and the changes on the database server are interleaved.

Constraint
A rule concerning the allowed values of attributes whose truth can be evaluated. A constraint usually does not include dynamic rules such as “Salesperson Pay can never decrease” or “Salary now must be greater than Salary last quarter.”

Database
A shared collection of logically related data (and a description of this data), designed to meet the information needs of an organization.

DBMS
A software system that enables users to define, create, and maintain the database and provides controlled access to this database.

Data dictionary
A user-accessible catalog of database and application metadata. An active data dictionary is a dictionary whose contents are updated automatically by the DBMS whenever changes are made to

the database or application structure. A passive data dictionary is one whose contents must be updated manually when changes are made.

Data integrity

The state of a database in which all constraints are fulfilled; usually refers to interrelation constraints in which the value of a foreign key must be present in the table having that foreign key as its primary key.

Deletion anomaly

In a relation, the situation in which the removal of one row of a table deletes facts about two or more themes.

Denormalization

The process of intentionally designing a relation that is not normalized. Denormalization is done to improve performance or security.

Determinant

One or more attributes that functionally determine another attribute or attributes. In the functional dependency (A, B) ? C, the attributes (A, B) are the determinant

Distributed database

A database that is stored and processed on two or more computers.

Domain

(1) The set of all possible values an attribute can have. (2) A description of the format (data type, length) and the semantics (meaning) of an attribute.

Entity

(1) Something of importance to a user that needs to be represented in a database. In an entity-relationship model, entities are restricted to things that can be represented by a single table.

File-based System

A collection of application programs that perform services for the end-users such as the production of reports. Each program defines and manages its own data.

Foreign key:

An attribute that is a key of one or more relations other than the one in which it appears.

Index

Overhead data used to improve access and sorting performance. Indexes can be constructed for a single column or groups of columns. They are especially useful for columns used for control breaks in reports and to specify conditions in joins.

Join

A relational algebra operation on two relations, A and B, that produces a third relation, C. A row of A is concatenated with a row of B to form a new row in C if the rows in A and B meet restrictions concerning their values. For example, A1 is an attribute in A, and B1 is an attribute in B. The join of A with B in which A1, B1 will result in a relation C having the concatenation of rows in A and B in which the value of A1 is less than the value of B1

Normalization

The process of evaluating a relation to determine whether it is in a specified normal form and, if necessary, of converting it to relations in that specified normal form.

Null value

An attribute value that has never been supplied. Such values are ambiguous and can mean (a) the value is unknown, (b) the value is not appropriate, or (c) the value is known to be blank.

Primary key

A candidate key selected to be the key of a relation.

Program/data independence

The condition existing when the structure of the data is not defined in application programs. Rather, it is defined in the database, and the application programs obtain it from the DBMS. In this way, changes can be made in the data structures that might not necessarily be made in the application programs.

Primary key

A candidate key selected to be the key of a relation.

Program/data independence

The condition existing when the structure of the data is not defined in application programs. Rather, it is defined in the database, and the application programs obtain it from the DBMS. In this way, changes can be made in the data structures that might not necessarily be made in the application programs.

Weak entity

In an entity-relationship model, an entity whose logical existence in the database depends on the existence of another entity. See ID-dependent entity and Strong entity.

References

Websites

- www.cs.usaka.ca
- www.cs.auckland.ac.nz
- www.cs.purdue.edu
- www.stanford.edu
- www.ocw.mit.edu
- www.w3schools.com
- www.wps.prenhall.com
- www.cs.sfu.ca

Books

- Thomas Connolly and Carolyn Begg, “Database Systems – A Practical Approach to Design, Implementation and Management” , Pearson Education , 4th Edition
- Elmasri and Navathe, “Fundamentals of Database Systems”, Pearson Education Edition, 4th Edition
- Korth, Sudarshan and Silberschatz, “Database System Concepts”, Mc Graw hill Publications, 5th Edition
- C J Date, “An Introduction to Database Management”, Addison Wesely Publications, 6th Edition.

STUDENT NOTES: