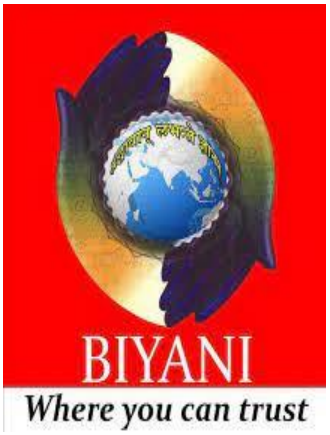
Biyani's Think Tank

Concept based notes

Database Management System

BCA II Sem.

Mr. Sachin Bagoria
Asst. Professor
Dept. of IT
Biyani Girls College, Jaipur



Published by :
Think Tanks
Biyani Group of Colleges

Sector-3, Vidhyadhar Nagar,
Jaipur-302 023 (Rajasthan)
Ph : 0141-2338371, 2338591-95 Fax : 0141-2338007
E-mail : acad@biyanicolleges.org
Website :www.gurukpo.com; www.biyanicolleges.org

ISBN : 978-93-83462-29-2

Edition: 2025

Leaser Type Setted by :
Biyani College Printing Department

Preface

I am glad to present this book, especially designed to serve the needs of the students. The book has been written keeping in mind the general weakness in understanding the fundamental concepts of the topics. The book is self-explanatory and adopts the “Teach Yourself” style. It is based on question-answer pattern. The language of book is quite easy and understandable based on scientific approach.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the readers for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chairman* & Dr. Sanjay Biyani, *Director* (*Acad.*) Biyani Group of Colleges, who are the backbones and main concept provider and also have been constant source of motivation throughout this Endeavour. They played an active role in coordinating the various stages of this Endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Author

UNIT- I

Database System Concepts & Architecture: Overview of DBMS, Basic DBMS terminology, data base system v/s file system, Advantages and dis-advantages of DBMS, Coded rules, data independence. Architecture of a DBMS, Schemas, Instances, Database Languages, Database Administrator, Data Models.

UNIT-II

Data Modeling: Data modeling using the Entity Relationship Model: ER model concepts, notation for ER diagram, mapping constraints, keys, Concepts of Super Key, candidate key, primary key, Generalization, aggregation.

Relational Model : Concepts, Constraints, Languages, Relational database design by ER & EER mapping, Relational algebra relational calculus. Relational Algebra, Fundamental operations of Relational Algebra.

UNIT –III

Database Design: Functional dependencies, loss less decomposition, Normalization : 1-NF, 2-NF,3-NF and BCNF. **Transaction Management :** Transactions: Concepts, ACID Properties, States Of Transaction, Serializaibility, Isolation, Checkpoints, Deadlock Handling.

Recovery System & Security : Failure Classifications, Recovery & Atomicity, Log Base Recovery, Recovery with Concurrent Transactions, Introduction to Security & Authorization.

UNIT-IV

Introduction to SQL: Characteristics of SQL, Advantages of SQL, SQL data types and literals, Types of SQL commands, SQL operators and their procedure, Tables, views and indexes, Queries and sub queries, Aggregate functions, insert, update and delete operations, Joins, Unions, Intersection, Minus in SQL.

Chapter-1

Data and Information

Q.1 What do you mean by Data and Information?

Ans.: Data are plain facts. The word "data" is plural for "datum." When data are processed, organized, structured or presented in a given context so as to make them useful, they are called Information. It is not enough to have data (such as statistics on the economy). Data themselves are fairly useless, but when these data are interpreted and processed to determine its true meaning, they becomes useful and can be named as Information.

Q.2 What do you mean by Database?

Ans.: Definitions of **Database** :

- An organized body of related information.
- In computing, a database can be defined as a structured collection of records or data that is stored in a computer so that a program can consult it to answer queries. The records retrieved in answer to queries become information that can be used to make decisions.
- An organized collection of records presented in a standardized format searched by computers.
- A collection of data organized for rapid search and retrieval by a computer.
- A collection of related data stored in one or more computerized files in a manner that can be accessed by users or computer programs via a database management system.
- An organized collection of information, data, or citations stored in electronic format that can be searched for specific information or records by techniques specific to each database.
- A logical collection of interrelated information, managed and stored as a unit, usually on some form of mass-storage system such as magnetic tape or disk.

- A database is a structured format for organizing and maintaining information that can be easily retrieved. A simple example of a database is a table or a spreadsheet.
- A database in an organized collection of computer records. The most common type of database consists of records describing articles in periodicals otherwise known as a periodical index.
- A database collects information into an electronic file, for example a list of customer addresses and associated orders. Each item is usually called a ‗record' and the items can be sorted and accessed in many different ways.
- A set of related files that is created and managed by a Database Management System (DBMS).
- A computerized collection of information.
- Integrated data files organized and stored electronically in a uniform file structure that allows data elements to be manipulated, correlated, or extracted to satisfy diverse analytical and reporting needs.
- A collection of information stored in one central location. Many times, this is the source from which information is pulled to display products or information dynamically on a website.
- Relational data structure used to store, query, and retrieve information.
- An organized set of data or collection of files that can be used for a specified purpose. A collection of interrelated data stored so that it may be accessed with user friendly dialogs.
- A large amount of information stored in a computer system.

Q.3 What are the basic objectives of the Database?

Ans.: A database is a collection of interrelated data stored with minimum redundancy to serve many users quickly and efficiently. The general objective is to make information access easy, quick, inexpensive, and flexible for the user. In data base design, several **specific objectives** can be considered as follows:

Controlled Redundancy

Ease of Learning and Use

Data Independence

Most Information in Low Cost

Accuracy and Integrity

Recovery from failure

Privacy and Security

Performance

Q.4 Define the Database Management System.

Ans.: (i) A **Database Management System** (DBMS), or simply a **Database System** (DBS), consists of :

- A collection of interrelated and persistent data (usually referred to as the **Database** (DB)).
- A set of application programs used to access, update and manage that data (which form the data Management System (MS)).

ĀĀ□Ā□ The goal of a DBMS is to provide an environment that is both **convenient** and **efficient** to use in :

- Retrieving information from the database.
- Storing information into the database.

Databases are usually designed to manage **large** bodies of information. This involves :

- Definition of structures for information storage (data modeling).
- Provision of mechanisms for the manipulation of information (file and systems structure, query processing).
- Providing for the safety of information in the database (crash recovery and security).
- Concurrency control if the system is shared by users.

Q.5 Describe the basic components of DBMS. Why do we need DBMS.

Or

What are the Advantages of DBMS over conventional file system.

Ans.: There are four basic components of Database Management System :

Data : Raw facts which we want to feed in the computer.

Hardware : On which the data to be processed.

Software : The interface between the hardware and user, by which the data will change into the information.

User : There are so many types of users some of them are application programmer, endcase users and DBA.

Purpose of Database Systems :

To see why database management systems are necessary, let's look at a typical ``File-Processing System'' supported by a conventional operating system.

The application is a savings bank :

- Savings account and customer records are kept in permanent system files.
- Application programs are written to manipulate files to perform the following **tasks** :

  Debit or credit an account.

  Add a new account.

  Find an account balance.

  Generate monthly statements.

Development of the System proceeds as follows :

- New application programs must be written as the need arises.
- New permanent files are created as required.
- but over a long period of time files may be in different formats, and
- Application programs may be in different languages.

So we can see there are problems with the Straight File-Processing Approach :

- **Data Redundancy and Inconsistency :**

  Same information may be duplicated in several places.

All copies may not be updated properly.

- **Difficulty in Accessing Data :**

  May have to write a new application program to satisfy an unusual request.

  E.g. find all customers with the same postal code.

  Could generate this data manually, but a long job.

- **Data Isolation :**

  Data in different files.

  Data in different formats.

  Difficult to write new application programs.

- **Multiple Users :**

  Want concurrency for faster response time.

  Need protection for concurrent updates.

  E.g. two customers withdrawing funds from the same account at the same time - account has $500 in it, and they withdraw $100 and $50. The result could be $350, $400 or $450 if no protection.

- **Security Problems :**

  Every user of the system should be able to access only the data they are permitted to see.

  E.g. payroll people only handle employee records, and cannot see customer accounts; tellers only access account data and cannot see payroll data.

  Difficult to enforce this with application programs.

- **Integrity Problems :**

  Data may be required to satisfy constraints.

  E.g. no account balance below $25.00.

  Again, difficult to enforce or to change constraints with the file-processing approach.

Above all problems lead to the development of **Database Management Systems**.

Advantages :

- An organized and comprehensiveness of recording the result of the firms activities.

- A receiver of data to be used in meeting the information requirement of the MIS users.
- Reduced data redundancy.
- Reduced updating errors and increased consistency.
- Greater data integrity and independence from applications programs.
- Improved data access to users through use of host and query languages.
- Improved data security.
- Reduced data entry, storage, and retrieval costs.
- Facilitated development of new applications program.
- Standard can be enforced: Standardized stored data format is particularly desirable as an old data to interchange or migration (change) between the system.
- Conflicting requirement can be handled.

Disadvantages :

- It increases opportunity for person or groups outside the organization to gain access to information about the firms operation.
- It increases opportunity for fully training person within the organization to misuse the data resources intentionally.
- The data approach is a costly due to higher H/W and S/W requirements.
- Database systems are complex (due to data independence), difficult, and time-consuming to design.
- It is not maintain for all organizations .It is only efficient for particularly large organizations.
- Damage to database affects virtually all applications programs.
- Extensive conversion costs in moving form a file-based system to a database system.
- Initial training required for all programmers and users.

Q.6 Define the Overall System Structure of Database Management System.

Ans.: Overall System Structure :

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

Components include :

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

In addition, several data structures are required for physical system implementation :

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

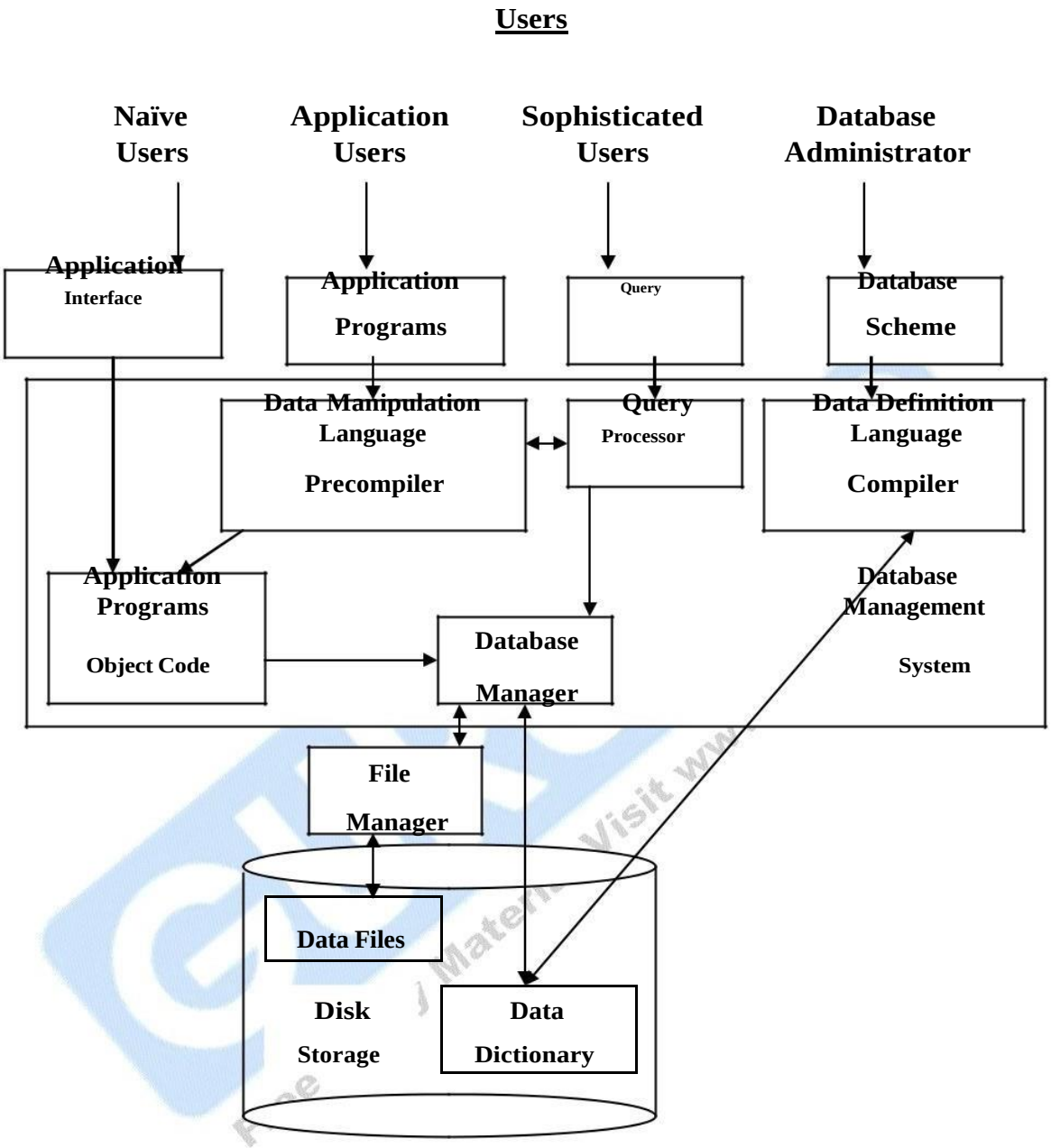


Figure : Database System Structure

□ □ □

Chapter-2

Database Architecture

Q.1 What do you mean by Data Abstraction?

Ans.: The major purpose of a database system is to provide users with an **abstract view** of the system. The system hides certain details of how data is stored, created and maintained. Complexity should be hidden from the database users, which is known as Data Abstraction.

Levels of Abstraction :

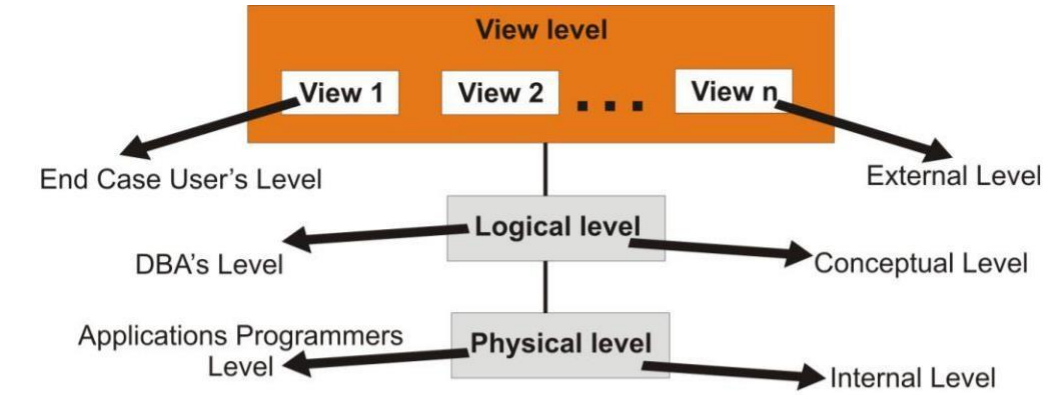
A common concept in computer science is *levels* (or less commonly *layers*) of abstraction, where in each level represents a different model of the same information and processes, but uses a system of expression involving a unique set of objects and compositions that are applicable only to a particular domain. Each relatively abstract, a "higher" level builds on a relatively concrete, "lower" level, which tends to provide an increasingly "granular" representation. For example, gates build on electronic circuits, binary on gates, machine language on binary, programming language on machine language, applications and operating systems on programming languages. Each level is embeded, but not determined, by the level beneath it, making it a language of description that is somewhat self-contained.

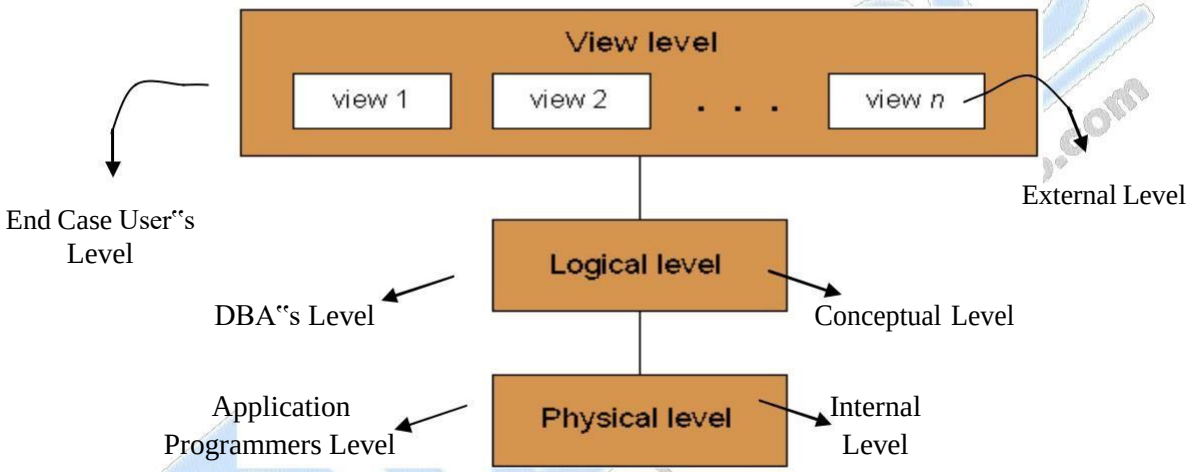
Q.2 Explain the Database Architecture with its various levels.

Ans.: Many users of database systems are 6 not deeply familiar

with computer

data structures, database developers often hide complexity through the following levels :





Physical Level : The lowest level of abstraction describes *how* the data is actually stored. The physical level describes complex low-level data structures in detail.

Logical Level : The next higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data. The logical level thus describes an entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical level structures, the user of the logical level does not need to be aware of this complexity. Database administrators, who must decide what information to keep in a database, use the logical level of abstraction.

View Level : The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity

remains because of the variety of information stored in a large database. Many users of a database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

Q.3 How will you define different Database Users in Database Management System?

Ans.: The **Database Users** fall into several categories :

- **Application Programmers** are computer professionals interacting with the system through DML calls embedded in a program written in a host language (e.g. C, PL/1, Pascal) :

  These programs are called **Application Programs**.

  The **DML Precompiler** converts DML calls (prefaced by a special character like $, #, etc.) to normal procedure calls in a host language.

  The host language compiler then generates the object code.

  Some special types of programming languages combine Pascal-like control structures with control structures for the manipulation of a database.

  These are sometimes called **Fourth-Generation Languages**.

  They often include features which to generate forms and display data.

- **Sophisticated Users** interact with the system without writing programs :

  They form requests by writing queries in a database query language.

  These are submitted to a query processor that breaks a DML statement down into instructions for the database manager module.

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

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

Chapter-3

Data Models

Q.1 Define various Data Models.

Ans.: Data Models: Data models are a collection of conceptual tools for describing data, data relationships, data semantics and data constraints. There are three different groups :

Object-Based Logical Models.

Record-Based Logical Models.

Physical Data Models.

We'll look at them in more detail now :

Object-Based Logical Models :

- Describe data at the conceptual and view levels.
- Provide fairly flexible structuring capabilities.
- Allow one to specify data constraints explicitly.
- Over 30 such models, including :

  Entity-Relationship Model

  Object-Oriented Model

  Binary Model

  Semantic Data Model

  Info Logical Model

  Functional Data Model

At this point, we'll take a closer look at the **Entity-Relationship (E-R)** and **Object-Oriented** models.

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

- An **entity** is a distinguishable object that exists.
- Each entity has associated with it a set of **attributes** describing it.
- E.g. *number* and *balance* for an account entity.
- A **relationship** is an association among several entities.
- E.g. A *cust_acct* relationship associates a customer with each account he or she has.
- The set of all entities or relationships of the same type is called the **entity set** or **relationship set**.
- Another essential element of the E-R diagram is the **mapping cardinalities**, which express the number of entities to which another entity can be associated via a relationship set.

The overall logical structure of a database can be expressed graphically by an **E-R diagram** :

- **Rectangles**: represent entity sets.
- **Ellipses**: represent attributes.
- **Diamonds**: represent relationships among entity sets.
- **Lines**: link attributes to entity sets and entity sets to relationships.

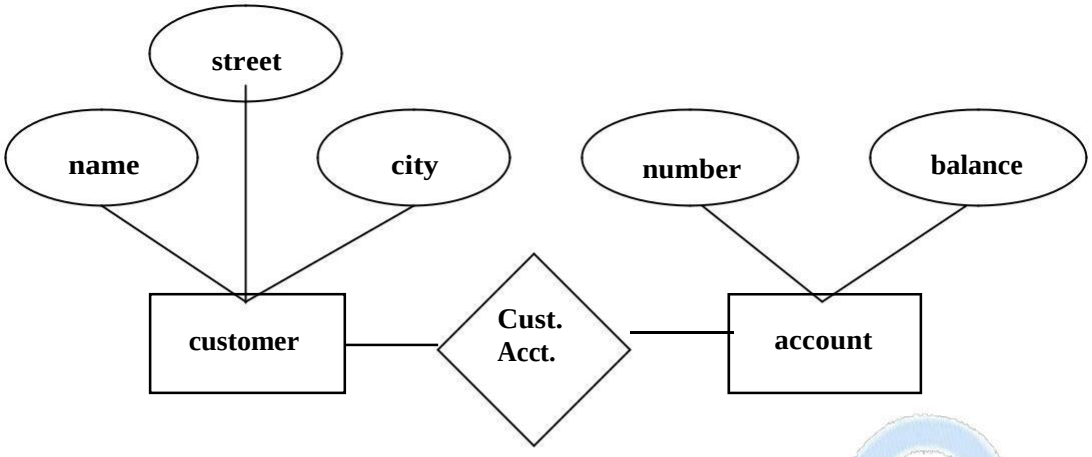


Figure : Entity-Relationship Model

Object-Oriented Model :

The object-oriented model is based on a collection of objects, like the E-R model.

- An object contains values stored in **instance variables** within the object.
- Unlike the record-oriented models, these values are themselves objects.
- Thus objects contain objects to an arbitrarily deep level of nesting.
- An object also contains bodies of code that operate on the object.
- These bodies of code are called **methods**.
- Objects that contain the same types of values and the same methods are grouped into **classes**.
- A class may be viewed as a type definition for objects.
- Analogy: the programming language concept of an abstract data type.
- The only way in which one object can access the data of another object is by invoking the method of that other object.
- This is called **sending a message** to the object.

- Internal parts of the object, the instance variables and method code, are not visible externally.
- Result is two levels of data abstraction.

For example, consider an object representing a bank account.

- The object contains instance variables number and balance.
- The object contains a method pay-interest which adds interest to the balance.
- Under most data models, changing the interest rate entails changing code in application programs.
- In the object-oriented model, this only entails a change within the pay-interest method.

Unlike entities in the E-R model, each object has its own unique identity, independent of the values it contains :

- Two objects containing the same values are distinct.
- Distinction is created and maintained in physical level by assigning distinct object identifiers

Record-Based Logical Models :

- Also describe data at the conceptual and view levels.
- Unlike object-oriented models, they are used to
  - Specify overall logical structure of the database, and
  - Provide a higher-level description of the implementation.
- Named so because the database is structured in fixed-format records of several types.
- Each record type defines a fixed number of fields, or attributes.
- Each field is usually of a fixed length (this simplifies the implementation).
- Record-based models do not include a mechanism for direct representation of code in the database.
- Separate languages associated with the model are used to express database queries and updates.

- The three most widely-accepted models are the **relational, network**, and **hierarchical**.
- This course will concentrate on the **relational** model.
- The **network** and **hierarchical** models are covered in appendices in the text.

The Relational Model :

- Data and relationships are represented by a collection of **tables**.
- Each **table** has a number of columns with unique names, e.g. *customer, account*.
- Following tabler shows a sample relational database.

name	street	city	number
Lowery	Maple	Queens	900
Shiver	North	Bronx	556
Shiver	North	Bronx	647
Hodges	Sidehill	Brooklyn	801
Hodges	Sidehill	Brooklyn	647

number	balance
900	55
556	10000
647	105366
801	10533

Figure : A Sample Relational Database.

The Network Model :

- Data are represented by collections of records.
- Relationships among data are represented by links.
- Organization is that of an **arbitrary graph**.
- Following figure shows a sample network database that is the equivalent figure of the relational database.

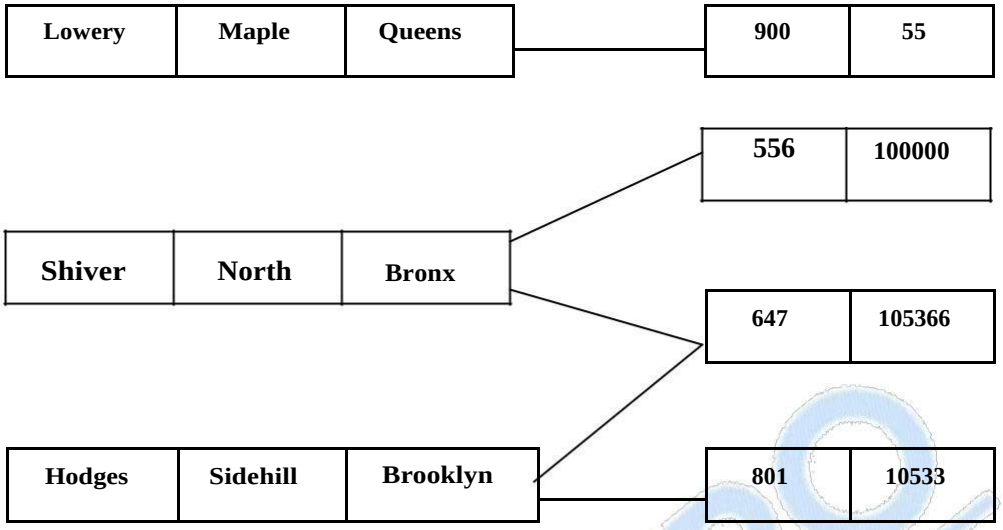


Figure : Sample Network

Database c) The Hierarchical Model :

- Similar to the network model.
- Organization of the records is as a collection of **trees**, rather than arbitrary graphs.
- Following figure shows a sample hierarchical database that is the equivalent figure of the relational database

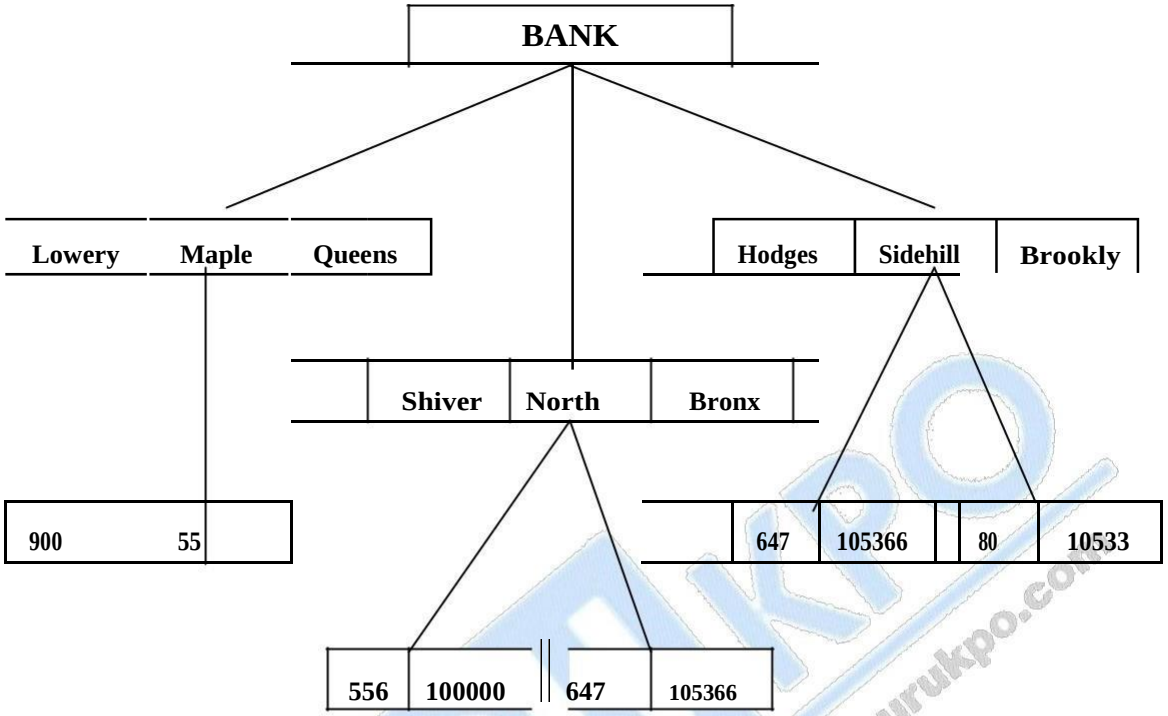


Figure : A Sample Hierarchical Database

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

Physical Data Models :

Are used to describe data at the lowest level.

Very few models, e.g.

- Unifying model
- Frame memory

Q.2 Define the components of Data Models.

Ans.: A data model underlines the structure of a database and defines the relationship between the different entities, a data model consist of three main components.

Structures : The structure portion specifies hoe elementary data item are grouped into layers.

Operation : The operation component provides mechanism for inaction relation, retrieval & modification of data

Constraints : The constraint define conditions that must be met for the data to be complete & correct. Constraints are used to control the condition under which a particular data object may exist be altered & do forth.

Three major types of constraints are on values depends & referential integrity :–

Constraints	**Description**
Values	The allowable, valid values for attribute may be States as a list, a range, type of character etc. For Values may be 1,2 or 3 only range from 0 to 60 or be 1,2 or 3 only range from 0 to 60 or be numeric only.
Dependencies	The allowable values for attribute may depend on it other values for the attribute may depend on it other values. For Ex. The Employee eligibility for overtime is depending upon his/her other status code.
Relational Integrity	Entity and relationship often have reference condition that must be met for e.g. there may be existence dependencies in which for one entity to exist. An illustration of this is sales order, for an order to exist there must be a customer.

Q.3 Explain IMS Hierarchy in terms of DBMS?

Ans.: IMS Hierarchy : IMS (Information Management System) is a database and transaction management system that was first introduced by IBM in 1968. Since then, IMS has gone through many changes in adapting to new programming tools and environments. IMS is one of two major legacy database and transaction management subsystems from IBM that run on mainframe MVS (now z/OS) systems. The other is CICS (Customer information service system). It is claimed that, historically, application programs that use either (or both) IMS or CICS services have handled and continue to handle most of the world's banking, insurance, and order entry transactions.

IMS consists of two major components, the IMS Database Management System (IMS DB) and the IMS Transaction Management System (IMS TM). In IMS DB, the data is organized into a hierarchy. The data in each level is dependent on the data in the next higher level. The data is arranged so that its integrity is ensured, and the storage and retrieval process is optimized. IMS TM controls I/O (input/output) processing, provides formatting, logging, and recovery of messages, maintains communications security, and oversees the scheduling and execution of programs. TM uses a messaging mechanism for queuing requests. IMS's original programming interface was DL/1 (Data Language/1). Today, IMS applications and databases can be connected to CICS applications and DB2 databases. Java programs can access IMS databases and services.

□□□

Chapter-4

Relational Algebra & Relational Calculus

Q.1 Define the following terms.

Ans.: (i) **Instances and Schemas :**

Databases change over time.

The information in a database at a particular point in time is called an **Instance** of the database.

The overall design of the database is called the database **Schema**.

Analogy with programming languages :

- Data type definition - Schema
- Value of a variable - Instance

There are several schemas, corresponding to levels of abstraction :

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

Data Independence :

The ability to modify a scheme definition in one level without affecting a scheme definition in a higher level is called **Data Independence**.

There are two kinds :

- **Physical Data Independence :**

  The ability to modify the physical scheme without causing application programs to be rewritten

  Modifications at this level are usually to improve performance

- **Logical Data Independence :**

  The ability to modify the conceptual scheme without causing application programs to be rewritten

  Usually done when logical structure of database is altered

Logical data independence is harder to achieve as the application programs are usually heavily dependent on the logical structure of the data. An analogy is made to abstract data types in programming languages.

Data Definition Language (DDL) :

Used to specify a database scheme as a set of definitions expressed in a DDL.

DDL statements are compiled, resulting in a set of tables stored in a special file called a data dictionary or data directory.

The data directory contains metadata (data about data).

The storage structure and access methods used by the database system are specified by a set of definitions in a special type of DDL called a **Data Storage and Definition** language

Basic Idea : hide implementation details of the database schemes from the users.

Data Manipulation Language (DML) :

Data Manipulation is : -

- **Retrieval** of information from the database.
- **Insertion** of new information into the database.
- **Deletion** of information in the database.
- **Modification** of information in the database.

A DML is a language which enables users to access and manipulate data.

The goal is to provide efficient human interaction with the system.

There are two types of DML :

- **Procedural** : The user specifies *what* data is needed and *how* to get it.

- **Nonprocedural** : The user only specifies *what* data is needed.

  Easier for user.

  May not generate code as efficient as that produced by procedural languages.

A **Query Language** is a portion of a DML involving information retrieval only. The terms DML and query language are often used synonymously.

Q.2 What do you mean by Database Manager and explain its responsibilities for DBMS.

Ans.: Database Manager :

The **Database Manager** is a **Program Module** which provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

Databases typically require lots of storage space (gigabytes). This must be stored on disks. Data is moved between disk and main memory (MM) as needed.

The goal of the database system is to **simplify** and **facilitate** access to data. Performance is important. Views provide simplification.

So the database manager module is responsible for :

- **Interaction with the File Manager :** Storing raw data on disk using the file system usually provided by a conventional operating system. The database manager must translate DML statements into low-level file system commands (for storing, retrieving and updating data in the database).
- **Integrity Enforcement :** Checking that updates in the database do not violate consistency constraints (e.g. no bank account balance below $25).
- **Security Enforcement :** Ensuring that users only have access to information they are permitted to see.
- **Backup and Recovery :** Detecting failures due to power failure, disk crash, software errors, etc., and restoring the database to its state before the failure.
- **Concurrency Control :** Preserving data consistency when there are concurrent users.

Some small database systems may miss some of these features, resulting in simpler database managers. (For example, no concurrency is required on a PC running MS-DOS.) These features are necessary on larger systems.

Q.3 Explain the concept of Relational Algebra.

Introduction :

Relational algebras received little attention until the publication of E.F. Codd's relational model of data in 1970. Codd proposed such an algebra as a basis for database query languages. The first query language to be based on Codd's algebra was ISBL, and this pioneering work has been acclaimed by many authorities as having shown the way to make Codd's idea into a useful language. Business System 12 was a short-lived industry-strength relational DBMS that followed the ISBL example. In 1998 Chris Date and Hugh Darwen proposed a language called **Tutorial D** intended for use in teaching relational database theory, and its query language also draws on ISBL's ideas. Rel is an implementation of **Tutorial D**. Even the query language of SQL is loosely based on a relational algebra, though the operands in SQL (tables) are not exactly relations and several useful theorems about the relational algebra do not hold in the SQL counterpart (arguably to the detriment of optimisers and/or users).

Because a relation is interpreted as the extension of some predicate, each operator of a relational algebra has a counterpart in predicate calculus. For example, the natural join is a counterpart of logical AND (). If relations *R* and *S* represent the extensions of predicates *p1* and *p2*, respectively, then the natural join of *R* and *S* (*R S*) is a relation representing the extension of the predicate *p1 p2*.

It is important to realise that Codd's algebra is not in fact complete with respect to first-order logic. Had it been so, certain insurmountable computational difficulties would have arisen for any implementation of it. To overcome these difficulties, he restricted the operands to finite relations only and also proposed restricted support for negation (NOT) and disjunction (OR). Analogous restrictions are found in many other logic-based computer languages. Codd defined the term ***relational completeness*** to refer to a language that is complete with respect to first-order predicate calculus apart from the restrictions he proposed. In practice the restrictions have no adverse effect on the applicability of his relational algebra for database purposes.

As in any algebra, some operators are primitive and the others, being definable in terms of the primitive ones, are derived. It is useful if the choice of primitive operators parallels the usual choice of primitive logical operators. Although it is well known that the usual choice in logic of AND, OR and NOT is somewhat arbitrary, Codd made a similar arbitrary choice for his algebra.

The six primitive operators of Codd's algebra are the *selection*, the *projection*, the *Cartesian product* (also called the *cross product* or *cross join*), the *set union*, the *set difference*, and the *rename*. (Actually, Codd omitted the rename, but the compelling case for its inclusion was shown by the inventors of ISBL.) These six operators are fundamental in the sense that none of them can be omitted without losing expressive power. Many other operators have been defined in terms of these six. Among the most important are set intersection, division, and the natural join. In fact ISBL made a compelling case for replacing the Cartesian product by the natural join, of which the Cartesian product is a degenerate case.

Altogether, the operators of relational algebra have identical expressive power to that of domain relational calculus or tuple relational calculus. However, for the reasons given in the Introduction above, relational algebra has strictly less expressive power than that of first-order predicate calculus without function symbols. Relational algebra actually corresponds to a subset of first-order logic that is Horn clauses *without* recursion and negation.

The **Relational Algebra** is a procedural query language.

- Six fundamental operations :

  Unary

  Select

  Project

  Rename

  Binary

  Cartesian product

  Union

  Set-difference

- Several other operations, defined in terms of the fundamental operations :

Set-intersection

Natural join

Division

Assignment

- Operations produce a new relation as a result.

There are some relations which we are using in our queries :

Account

account_number	branch_name	balance
A-101	Downtown	500
A-102	SFU	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Laugheed Mall	700
A-305	Round Hill	350

(ii) Branch

branch_name	branch-city	assets
Brighton	Brooklyn	7100000
Downtown	Vaneouver	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
SFU	Burnaby	1700000
Pownal	Bennington	300000
Lougheed Mall	Burnaby	2100000
Round Hill	Horseneck	8000000

(iii) Customer

customer_name	customer_street	customer_city
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye

Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
customer_name	**customer_street**	**customer_city**
Hayes	Main	Harrison
Johnson	Alma	PaloAlto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

(iv) Depositor

customer-name	account-number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

(v) Loan

Loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downton	1500
L-15	SFU	1500
L-16	SFU	1300
L-17	Downtown	1000
L-23	Laugheed Mall	2000
L-93	Mianus	500

(vi) Borrower

customer_name	loan-number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

Fundamental Operations :

The Select Operation :

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

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

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

Let Figure be the *borrow* and *branch* relations in the banking example :

bname	loan#	ename	amount
Downtown	17	Jones	1000
Lougheed Mall	23	Smith	2000
SFU	15	Hayes	1500

bname	assets	bcity
Downtown	9,00,000	Vaneouver
Lougheed Mall	21,000,000	Burnaby
SFU	17,000,000	Burnaby

Figure : The *borrow* and *branch* Relations

The new relation created as the result of this operation consists of one tuple :

(SFU, 15, Hayes, 1500)

We allow comparisons using **=, ≠, <, ≤, > and ≥** in the selection predicate.

We also allow the logical connectives ∨ (or) and ∧ (and). For example :

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

ename	banker
Hayes	Jones
Johnson	Johnson

Figure : The *client* Relation

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

Client_scheme = (ename, banker)

we might write

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

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

The Project Operation :

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

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

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

$\prod_{bname, ename}(borrow)$

We can perform these operations on the relations resulting from other operations.

To get the names of customers having the same name as their bankers,

$\prod_{ename}(\sigma_{ename = banker}(client))$

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

The Cartesian Product Operation :

The **Cartesian Product** of two relations is denoted by a cross (**x**), written

r_1 x r_2 for relations r_1 and r_2

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

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

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

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

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

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

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

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

$$\sigma_{client.ename = customer.ename}(\sigma_{banker = \text{“Johnson”}}(client \times customer))$$

to get just these tuples.

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

$$\prod_{client.ename,\ ecity}(\sigma_{client.ename = customer.ename}(\sigma_{banker = \text{“Johnson”}}(client \times customer)))$$

The Rename Operation :

The **Rename Operation** solves the problems that occurs with naming when performing the Cartesian Product of a relation with itself.

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

We can get the street and city of Smith by writing

$$\prod_{street,\ ecity}(\sigma_{ename = \text{“Smith”}}(customer))$$

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

$$\sigma_p(customer \times (\prod_{street,\ ecity}(\sigma_{ename = \text{“Smith”}}(customer))))$$

where **P** is a selection predicate requiring *street* and *ecity* values to be equal.

Problem: how do we distinguish between the two street values appearing in the Cartesian product, as both come from a *customer* relation?
Solution: use the rename operator, denoted by the Greek letter rho (**ρ**).

We write **ρz(r)**

to get the relation **r** under the name of **z**.

If we use this to rename one of the two **customer** relations we are using, the ambiguities will disappear.

$$\prod \text{customer . ename } (\sigma_{\text{cust2 . street = customer . street } \wedge \text{ cust2 . ecity = customer . ecity}}$$

$$\text{(customer x } (\prod \text{street , ecity } (\sigma_{\text{name = "Smith"}} (\rho_{\text{cust2}}(\text{customer})))))$$

The Union Operation :

The **Union Operation** is denoted by ∪ as in set theory. It returns the union (set union) of two compatible relations.

For a union operation **r U s** to be legal, we require that

- **r** and **s** must have the same number of attributes.
- The domains of the corresponding attributes must be the same.

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

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

$$\prod_{\text{ename}}(\sigma_{\text{balance = "SFU"}}(\text{borrow})) \cup \prod_{\text{ename}}(\sigma_{\text{bname = "SFU"}}(\text{deposit}))$$

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

(a)

ename
Hayes
Adams

(b)

ename
Adams

Figure : The *union* and *set-difference* operations

The Set Difference Operation

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

Thus r — s results in a relation containing tuples that are in **r** but not in **s**

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

$$\prod_{ename}(\sigma_{balance = \text{“SFU”}}(deposit)) - \prod_{ename}(\sigma_{bname = \text{“SFU”}}(borrow))$$

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

Strategy :

Find a relation **r** containing the balances **not** the largest.

Compute the set difference of **r** and the *deposit* relation. To

find **r**, we write

$$\prod_{deposit.balance}(\sigma_{deposit.balance < d.balance}(deposit \times \rho_{d}(deposit)))$$

This resulting relation contains all balances except the largest one.

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

$$\prod_{balance}(deposit) - \prod_{deposit.balance}(\sigma_{deposit.balance < d.balance}(deposit \times \rho_{d}(deposit)))$$

Figure shows the result.

(a)

balance
400
500
700

(b)

balance
1300
900

Figure 1 : Find the largest account balance in the bank

Formal Definition of Relational Algebra :

A basic expression consists of either

- A relation in the database.
- A constant relation.

General expressions are formed out of smaller subexpressions using

- $\sigma_P(E_1)$ select (p a predicate)
- $\prod_S(E_1)$ project (s a list of attributes)
- $\rho_Z(E_1)$ rename (x a relation name)

$E_1 \cup E_2$ union

$E_1 - E_2$ set difference

$E_1 \times E_2$ Cartesian product

Additional Operations :

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

The Set Intersection Operation

Set intersection is denoted by ∩, and returns a relation that contains tuples that are in **both** of its argument relations.

It does not add any power as

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

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

$$\prod_{ename}(\sigma_{bname = \text{"SFU"}}(borrow)) \cap \prod_{ename}(\sigma_{bname = \text{"SFU"}}(deposit))$$

The Natural Join Operation :

Often we want to simplify queries on a Cartesian product.

For example, to find all customers having a loan at the bank and the cities in which they live, we need *borrow* and *customer* relations :

$$\prod_{borrow.ename,\ ecity}(\sigma_{borrow.ename = customer.ename}(borrow \times customer))$$

Our selection predicate obtains only those tuples pertaining to only one *cname*.

This type of operation is very common, so we have the **natural join**, denoted by a **x** sign. Natural join combines a cartesian product and a selection into one operation. It performs a selection forcing equality on those attributes that appear in both relation schemas. Duplicates are removed as in all relation operations.

To illustrate, we can rewrite the previous query as

$$\prod_{ename,\ ecity}(borrow \times customer))$$

The resulting relation is shown in Figure.

ename	ecity
Smith	Burnaby

Hayes	Burnaby
Jones	Vancouver

Figure : Joining *borrow* and *customer* relations

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

Consider **R** and **S** to be **sets** of attributes.

We denote attributes appearing in **both** relations by **R ∩ S**.

- We denote attributes in **either or both** relations by **R** U **S**.
- Consider two relations **r(R)** and **s(S)**.

The natural join of **r** and **s**, denoted by **r x s** is a relation on scheme **R** U **S**.

It is a projection onto **R** U **S** of a selection on **r x s** where the predicate requires **r . A = s . A** for each attribute **A** in **R ∩ S**.

Formally,

$$r \times s = \prod_{R \cup S}(\sigma_{r.A1 = s.A1 \wedge r.A2 = s.A2 \wedge \ldots \wedge r.An = s.An}(r \times s))$$

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

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

$$\prod_{bname , assets}(\sigma_{city = \text{“Stamford”}} (customer \times deposit \times branch))$$

Note that **x** is associative.

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

$$\prod_{ename}(\sigma_{bname = \text{“SFU”}} (borrow \times deposit))$$

This is equivalent to the set intersection version we wrote earlier. We see now that there can be several ways to write a query in the relational algebra.

If two relations **r(R)** and **s(S)** have no attributes in common, then **R ∩ S = 0**, and **r x s = s x r.**

The Division Operation :

Division, denoted by ÷, is suited to queries that include the phrase ``for all‖.

Suppose we want to find all the customers who have an account at all branches located in Brooklyn.

Strategy: think of it as three steps.

We can obtain the names of all branches located in Brooklyn by

$$r_1 = \prod_{bname} (\sigma_{bcity = \text{“Brooklyn”}} (branch))$$

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

$$r_2 = \prod_{ename,\ bname}(deposit)$$

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

The divide operation provides exactly those customers:

$$\prod_{ename,\ bname}(deposit) \div \prod_{bname} (\sigma_{city = \text{“Brooklyn”}} (branch))$$

which is simply $r_2 \div r_1$.

Formally,

Let **r(R)** and **s(S)** be relations.

- Let $S \subseteq R$.

The relation $r \div s$ is a relation on scheme $R - S$.

A tuple t is in $r \div s$ if for every tuple t_s in s there is a tuple t_r in r satisfying both of the following:

$$t_r[S] = t_s[S]$$

$$t_r[R - S] = t_s[R - S]$$

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

The division operation can be defined in terms of the fundamental operations.

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

The Assignment Operation

Sometimes it is useful to be able to write a relational algebra expression in parts using a temporary relation variable (as we did with r_1 and r_2 in the division example).

The assignment operation, denoted ← works like assignment in a programming language.

We could rewrite our division definition as

temp ← ∏R -S(r)

temp ← ∏R -S((temp x s) – r)

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

Q.4 Explain the term Relational Calculus in DBMS.

Ans.: The Tuple Relational Calculus :

The tuple relational calculus is a nonprocedural language. (The relational algebra was procedural.)

We must provide a formal description of the information desired.

A query in the tuple relational calculus is expressed as

{ t | P(t) }

i.e. the set of tuples **t** for which predicate **P** is true.

We also use the notation

- **t[a]** to indicate the value of tuple **t** on attribute **a**.

t Є r to show that tuple **t** is in relation **r**.

Example Queries

For example, to find the branch-name, loan number, customer name and amount for loans over $1200:

{ t | t Є borrow ∧ t[amount] > 1200 }

This gives us all attributes, but suppose we only want the customer names. (We would use **project** in the algebra.)

We need to write an expression for a relation on scheme *(cname)*.

{ t | ∃δ t Є borrow (t[ename] = s[ename] ∧ s[amount] > 1200)}

In English, we may read this equation as ``the set of all tuples **t** such that there exists a tuple **s** in the relation *borrow* for which the values of **t** and **s** for the *cname* attribute are equal, and the value of **s** for the *amount* attribute is greater than 1200.''

The notation $\exists t \in r(Q(t))$ means ``there exists a tuple **t** in relation **r** such that predicate **Q(t)** is true''.

How did we get the above expression? We needed tuples on scheme *cname* such that there were tuples in *borrow* pertaining to that customer name with *amount* attribute > **1200.**

The tuples get the scheme *cname* implicitly as that is the only attribute it is mentioned with.

Let's look at a more complex example.

Find all customers having a loan from the SFU branch, and the the cities in which they live:

$$\{ t \mid \exists s \in borrow (t[ename] = s[ename] \wedge s[amount] = \text{“SFU”})\}$$

$$\{ \wedge \exists u \in customer (u[ename] = s[ename] \wedge t[ecity] = u[ecity])\}$$

In English, we might read this as ``the set of all *(cname,ccity)* tuples for which *cname* is a borrower at the SFU branch, and *ccity* is the city of *cname*''.

Tuple variable **s** ensures that the customer is a borrower at the SFU branch.

Tuple variable **u** is restricted to pertain to the same customer as **s**, and also ensures that *ccity* is the city of the customer.

The logical connectives $\wedge$(AND) and $\vee$(OR) are allowed, as well as $\neg$ (negation).

We also use the existential quantifier $\exists$ and the universal quantifier $\forall$.

Some more examples :

Find all customers having a loan, an account, or both at the SFU branch :

$$\{ t \mid \exists s \in borrow (t[ename] = s[ename] \wedge s[bname] = \text{“SFU”})\}$$

$$\{ \wedge \exists u \in deposit (t[ename] = u[ename] \wedge u[bname] = \text{“SFU”})\}$$

Note the use of the $\vee$ connective.

As usual, set operations remove all duplicates.

Find all customers who have **both** a loan and an account at the SFU branch.

Solution: simply change the $\vee$ connective in 1 to a $\wedge$.

Find customers who have an account, but **not** a loan at the SFU branch.

{ t | ∃u t Є deposit (t[ename] = u[ename] ∧ u[bname] = "SFU")}

{ A — ∃s Є borrow (t[ename] = s[ename] ∧ s[bname] = "SFU")}

Find all customers who have an account at **all** branches located in Brooklyn. (We used **division** in relational algebra.)

For this example we will use implication, denoted by a pointing finger in the text, but by here.

The formula **P ⇒ Q** means **P** implies **Q**, or, if **P** is true, then **Q** must be true.

{ t | ∀u Є brach (u[city] = "Brooklyn" ⇒

(∃s Є deposit (t[ename] = s[ename] ∧ u[bname] = s[bname]))}

In English: the set of all *cname* tuples **t** such that for all tuples **u** in the *branch* relation, if the value of **u** on attribute *bcity* is Brooklyn, then the customer has an account at the branch whose name appears in the *bname* attribute of **u**.

Formal Definitions :

A tuple relational calculus expression is of the form

{ t | P(t) }

where **P** is a **formula**.

Several tuple variables may appear in a formula.

A tuple variable is said to be a **free variable** unless it is quantified by a ∃ or ∀. Then it is said to be a **bound variable**.

A formula is built of **atoms**. An atom is one of the following forms:

- **s Є r**, where s is a tuple variable, and r is a relation (is not allowed).
- **s[x] u[y]**, where s and u are tuple variables, and x and y are attributes, and is a comparison operator **(<, , =, ≤ , >, ≥).**
- **s[x]** θ **c,** where c is a constant in the domain of attribute **x**.

Formulae are built up from atoms using the following rules :

- An atom is a formula.

- If **P** is a formula, then so are $\neg$**P and (P)**.
- If P_1 and P_2 and are formulae, then so are $P_1 \wedge P_2$, $P_1 \vee P_2$ and $P_1 \Rightarrow P_2$.

If **P(s)** is a formula containing a free tuple variable **s**, then

$\exists$**s** $\in$ **r(P(s)) and** $\forall$ **s** $\in$ **r(P(s))** are formulae also.

Note some equivalences : -

- $P_1 \wedge P_2 = \neg(\neg P_1 \vee \neg P_2)$
- $\forall t \in r(P(t)) = \exists t \in r(\neg P(t))$
- $P_1 => P_2 = \neg P_1 \vee P_2$

The Domain Relational Calculus :

Domain variables take on values from an attribute's domain, rather than values for an entire tuple. Formal Definitions

An expression is of the form

$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n) \}$

where the x_{i1} $1 \leq i \leq n_1$ represent domain variables, and **P** is a **formula**.

An atom in the domain relational calculus is of the following forms

$< x_1, x_2, \ldots, x_n > \in$ **r** where **r** is a relation on **n** attributes, and x_{i1} $1 \leq i \leq n_1$, are domain variables or constants.

- **x** θ **y,** where **x** and **y** are domain variables, and θ is a comparison operator.

  x θ **c**, where c is a constant.

Formulae are built up from atoms using the following rules :

An atom is a formula.

- If **P** is a formula, then so are $\neg$**P** and **(P)**.
- If P_1 and P_2 are formulae, then so are $P_1 \wedge P_2$, $P_1 \vee P_2$ and $P_1 \Rightarrow P_2$.
- If **P(x)** is a formula where **x** is a domain variable, then so are $\exists$**x (P(x))** and $\forall$**x P(x))** .

Example Queries :

Find branch name, loan number, customer name and amount for loans of over $1200.

$$\{ < b_1\ I_1\ c_1\ a > \mid b_1\ I_1\ c_1\ a \in borrow \wedge a > 1200 \}$$

Find all customers who have a loan for an amount > than $1200.

$$\{ < c > \mid \exists b_1\ I_1\ a\ (< b_1\ I_1\ c_1\ a > \in borrow \wedge a > 1200)\}$$

Find all customers having a loan from the SFU branch, and the city in which they live.

$$\{ < c_1\ x > \mid \exists b_1\ I_1\ a\ (< b_1\ I_1\ c_1\ a > \in borrow$$

$$\wedge\ b = \text{“SFU”} > \ \wedge\ \exists y\ (\ c_1\ y_1\ x > \in customer\ ))\}$$

Find all customers having a loan, an account or both at the SFU branch.

$$\{ < c > \mid \exists b1\ I1\ a\ (< b1\ I1\ c1\ a > \in borrow \wedge b = \text{“SFU”})$$

$$V\ \exists\ b1\ a1\ n\ (< b1\ a1\ c1\ a > \in deposit \wedge b = \text{“SFU”})\}$$

Find all customers who have an account at **all** branches located in Brooklyn.

$$\{ < c > \mid \forall x1\ y1\ z\ (\ \neg (\ < x1\ y1\ z > \in branch)$$

$$\vee\ z \neq \text{”Brooklyn”}\ b1\ a1\ n\ \vee\ (\ \exists a1\ n\ (< x1\ a1\ c1\ n > \in deposit\ )))\}$$

If you find this example difficult to understand, try rewriting this expression using implication, as in the tuple relational calculus example. Here's my attempt:

$$\{ < cn > \mid \forall bn1\ as1\ bc\ (\ (\ < bn1\ as1\ bc > \in branch)$$

$$\vee\ bc \neq \text{”Brooklyn”}) \Rightarrow \exists an1\ ba\ (< cn1\ an1\ ba1\ bn > \in deposit\ )))\}$$

I've used two letter variable names to get away from the problem of having to remember what **x** stands for.

□ □ □

Chapter-5

Concept of Network

Q.1 Explain the concept of Network Model with its implementation.

Ans.: The Network Model :

The network data model was formalized in the late 1960s by the Database Task Group of the Conference on Data System Language (DBTG/ CODASYL). Their first report, which has been, revised a number of times, contained detailed specifications for the network data model (a model conforming to these specifications is also known as the DBTG data model). The specifications contained in the report and its subsequent revisions have been subjected to much debate and criticism. Many of the current database applications have been built on commercial DBMS systems using the DBTG model.

Implementation of the Network Data Model :

The record is a basic unit to represent data in the DBTG network database model. The implementation of the one-to-many relationships of a set is represented by linking the members of a given occurrence of a set to the owner record occurrence. The actual method of linking the member record occurrence to the owner is immaterial to the user of the database; however, here, we can assume that the set is implemented using a linked list. The list starts at the owner record occurrence and links all the member record occurrences with the pointer in the last member record occurrence leading back to the owner record. Note that for simplicity we have shown only one of the record fields of each record. This method of implementation assigns one pointer (link) in each record for each set type in which the record participates and, therefore, allows a record occurrence to participate in only one occurrence of a given set type. Any other method of implementing the set construct in a database management system based on the DBTG proposal is, in effect, equivalent to the linked list method.

A second form of network implementation, especially useful for M:N relationships, is a bit map. A bit map is a matrix created for each relationship. Each row corresponds to the relative record number of a target record of a relationship. A 1 bit in a cell for row X and column Y means that the records corresponding to row X and column Y are associated in this relationship; a zero means no association. For example, the PRODUCT with relative record number X is related to VENDOR with relative record numbers 1 and Y (and possibly others not shown). Bit maps are powerful data structures for the following reasons -

- Any record type(s) can be included in rows or columns.
- 1:1, 1:M, and M:1 relationships can all be represented.
- Rows and columns can be logically manipulated by Boolean operators ("and," "or," "not") to determine records that satisfy complex associations (e.g., any record that has both parent S and parent T).
- A bit map can be manipulated equally as well in either a row or column access (all the row records for a common column or all the column records for a common row) and can be easily extended for n-ary relationships).

Q.2 Define DBTG Network Model.

Ans.: DBTG Model :

The DBTG model uses two different data structures to represent the database entities and relationships between the entities, namely record type and set type. A record type is used to represent an entity type. It is made up of a number of data items that represent the attributes of the entity.

A set is used to represent a directed relationship between two record types, the so-called owner record type, and the member record type. The set type, like the record type, is named and specifies that there is a one-to-many relationship (I:M) between the owner and member record types. The set type can have more than one record type as its member, but only one record type is allowed to be the owner in a given set type. A database could have one or more occurrences of each of its record and set types. An occurrence of a set type consists of an occurrence of the owner record type and any number of occurrences of each of its member record types. A record type cannot be a member of two distinct occurrences of the same owner set type.

Bachman introduced a graphical means called a data structure diagram to denote the logical relationship implied by the set. Here a labeled rectangle represents the corresponding entity or record type. An arrow that connects two labeled rectangles represents a set type. The arrow direction is from the owner record type to the member record type. Figure shows two record types (DEPARTMENT and EMPLOYEE) and the set type DEPT-EMP, with DEPARTMENT as the owner record type and EMPLOYEE as the member record type.

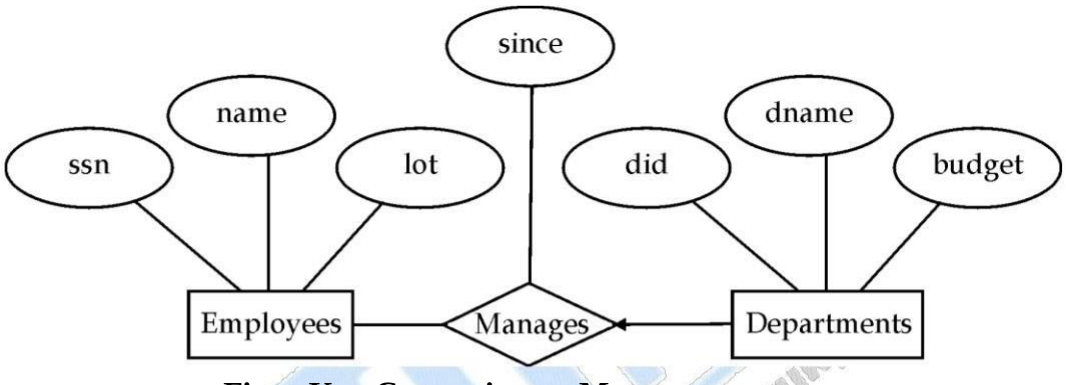


Fig. : Key Constraint on Manages

The data structure diagrams have been extended to include field names in the record type rectangle, and the arrow is used to clearly identify the data fields involved in the set association. A one-to-many (1:M) relationship is shown by a set arrow that starts from the owner field in the owner record type. The arrow points to the member field within the member record type. The fields that support the relationship can be clearly identified.

Q. 3. How can we convert an E-R diagram into a network database.

Ans. Each entity type in an E-R diagram is represented by a logical record type with the same name. The attributes of the entity are represented by data fields of the record. We use the term logical record to indicate that the actual implementation may be quite different.

The conversion of the E-R diagram into a network database consists of converting each 1:M binary relationship into a set (a 1:1 binary relationship being a special case of a 1:M relationship). If there is a 1:M binary relationship R1 from entity type E1 to entity type E2,

then the binary relationship is represented by a set An instance of this would be S1 with an instance of the record type corresponding to entity E1 as the

owner and one or more instances of the record type corresponding to entity E2 as the member. If a relationship has attributes, unless the attributes can be assigned to the member record type, they have to be maintained in a separate logical record type created for this purpose. The introduction of this additional record type requires that the original set be converted into two symmetrical sets, with the record corresponding to the attributes of the relationship as the member in both the sets and the records corresponding to the entities as the owners.

Each many-to-many relationship is handled by introducing a new record type to represent the relationship wherein the attributes, if any, of the relationship are stored. We then create two symmetrical 1:M sets with the member in each of the sets being the newly introduced record type.

In the network model, the relationships as well as the navigation through the database are predefined at database creation time.

Q.4 What do you mean by Database Administrator and explain its duties? Also define DBA Schema.

Ans.: Database Administrator :

The **Database Administrator** is a **person** having central control over data and programs accessing that data. Duties of the database administrator include :

- **Schema Definition :** the creation of the original database scheme. This involves writing a set of definitions in a DDL (data storage and definition language), compiled by the DDL compiler into a set of tables stored in the data dictionary.
- **Storage Structure and Access Method Definition :** writing a set of definitions translated by the data storage and definition language compiler
- **Scheme and Physical Organization Modification :** writing a set of definitions used by the DDL compiler to generate modifications to appropriate internal system tables (e.g. data dictionary). This is done rarely, but sometimes the database scheme or physical organization must be modified.
- **Granting of Authorization for Data Access :** granting different types of authorization for data access to various users

- **Integrity Constraint Specification :** generating integrity constraints. These are consulted by the database manager module whenever updates occur.

Q.5 What are the Responsibilities or Functions of Database Administrator?

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

Recoverability : Creating and testing Backups.

Integrity : Verifying or helping to verify data integrity.

Security : Defining and/or implementing access controls to the data.

Availability : Ensuring maximum uptime.

Performance - Ensuring maximum performance.

Development and Testing Support : Helping programmers and engineers to efficiently utilize the database.

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

□ □ □

Chapter-6

Types of Databases

Q.1 Define ORDBMS and OODBMS.

Ans.: An **Object-Relational Database** (**ORD**) or **Object-Relational Database Management System** (**ORDBMS**) is a database management system similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, it supports extension of the data model with custom data-types and methods.

One aim for this type of system is to bridge the gap between conceptual datamodeling techniques such as ERD and ORM, which often use classes and inheritance, and relational databases, which do not directly support them.

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

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

An object-relational database can be said to provide a middle ground between relational databases and *object-oriented databases* (OODBMS). In object-relational databases, the approach is essentially that of relational databases: the data resides in the database and is manipulated collectively with queries in a query language; at the other extreme are OODBMSes in which the database is essentially a persistent object store for software written

in an object-oriented programming language, with a programming API for storing and retrieving objects, and little or no specific support for querying.

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

Q.2 What are the fundamental characteristics of Distributed Databases?

Ans.: DDBMS have following characteristics :

- A collection of logically related shared data.
- The data is split into number of fragments.
- Fragments may be replicated.
- Fragments/replicas are allocated to sites.
- The sites are linked with computer network.
- The data at each site is under the control of a DBMS.
- The DBMS at each site can handle local applications autonomously.
- Each DBMS participates in at least one global application.
- It is not necessary for every site in the system to have its own local database as shown.
- The system is expected to make the distribution ***transparent*** to the user.
- Distributed database is split into fragments that can be stored on different computers and perhaps replicated.
- The objective of the transparency is to make the distributed system to appear like a centralized system.
- The system consists of data that is physically distributed across the network. If the data is centralized, even though the users may be accessing the data over the network, it is not considered as distributed DBMS, simply *distributed processing.*

Q.3 Write the different advantages and disadvantages of DDBMS.

Ans.: Advantages :

- Reflects organizational structure
- Improved sharability and local autonomy
- Improved availability
- Improved reliability
- Improved performance
- Modular growth
- Less danger on single-point failure

Disadvantages:

- Complexity
- Cost
- Security
- Integrity control more difficult
- Lack of standards
- Lack of experience
- Database design more complex
- Possible slow response

Q.4 Explain the different types of DDBMS.

Ans: There are two types of DDBMS : **Homogeneous** and **Heterogeneous DDBMSs.**

Homogeneous DDBMS :

- In homogeneous DDBMS, all sites use the same DBMS product.
- Much easier to design and manage.
- This design provides incremental growth by making additional new sites to DDBMS easy.
- Allows increased performance by exploiting the parallel processing capability of multiple sites.

Heterogeneous DDBMS :

- In heterogeneous DDBMS, all sites may run different DBMS products, which need not to be based on the same underlying data model and so

the system may be composed of RDBMS, ORDBMS and OODBMS products.

- In heterogeneous system, communication between different DBMS is required for translations.
- In order to provide DBMS transparency, users must be able to make requests in the language of the DBMS at their local site.
- Data from the other sites may have different hardware, different DBMS products and combination of different hardware and DBMS products.
- The task for locating those data and performing any necessary translation are the abilities of heterogeneous DDBMS.

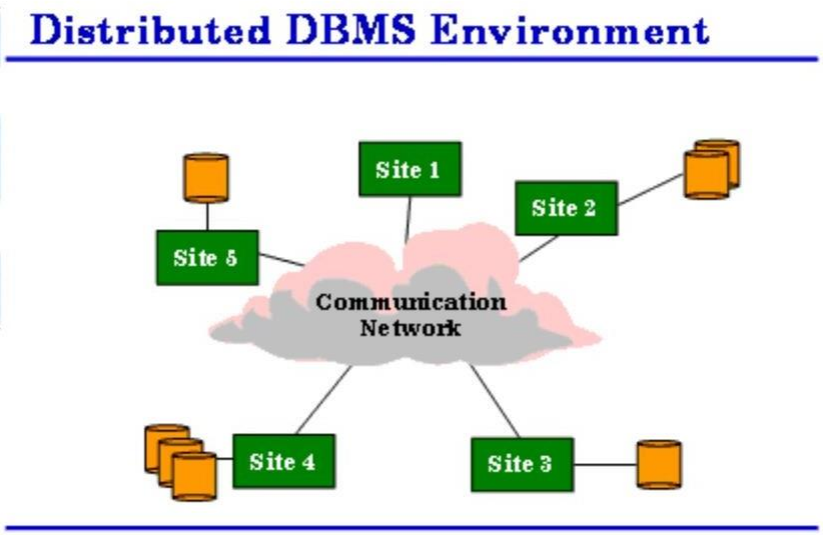
Q.5 What is not a DDBMS?

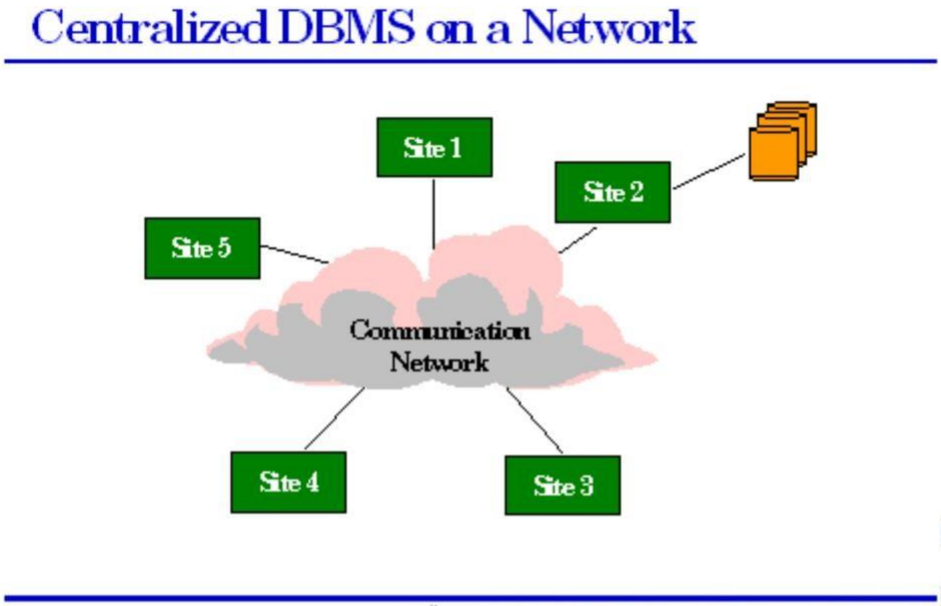
or

What do you mean by centralized database.

Ans.: The following are not a DDBMS :

- A time sharing computer system.
- A loosely or tightly coupled multiprocessor system.
- A database which resides at one of the nodes of a network of computers – this is a centralized database on a network node.





•

Q.4 What do you mean by Centralized Databases?

Ans.: A "centralized DBMS"is a DBMS where all the data within the database is stored on a single computer, usually the secondary storage device of the computer. In a centralized DBMS, as the number of transactions executing on the DBMS increases, performance of the DBMS significantly decreases and becomes a drain on the overall performance of the computer

Q.5 How the Distributed Databases is different from Centralized Databases?

Ans.: A database which resides all of its data centralized and on one machine or some machines which are connected together but looks one for the users looking from outside. And whoever wants to use data, he picks data from there and uses and saves again there.

While Distributed databases can be defined as a collection of multiple, logically interrelated databases distributed over a computer network.

And distributed database management system (DDBMS) manages the distributed databases and makes this distribution transparent to the user.

All the database must be logically related that are managed by DDBMS (distributed database management system). The distributed databases are not just the _collection of files' stored individually at different network nodes. Rather to form DDBS (distributed databases) all the files should be logically related and there should be structures among those files.

In the case of distributed databases, data must be physically distributed across the network nodes otherwise they will simply be separate databases not ‗distributed databases'.

Sometimes the multiprocessor system is also considered as distributed data processing systems but in fact it is not true.

As multiprocessor system may use either ‗shared-nothing architecture' or ‗shared-everything architecture'. Shared-nothing architecture system may work like distributed computing environment but it is not. because in distributed environment there may be heterogeneity of hardware as well as operating system at different sites in network which is not the case in multiprocessor systems. Thus for distributed databases data must be distributed over network nodes.

While centralized databases are managed by DBMS, and no data distribution is done in this case.

Distributed data is defined as collection of logically distributed database which are connected with each other through a network. A distributed database management system is used for managing distributed database. Each side has its own database and operating system.

A centralized database has all its data on one place. As it is totally different from distributed database which has data on different places. In centralized database as all the data reside on one place so problem of bottle-neck can occur, and data availability is not efficient as in distributed database. While in distributed databases users can issue commands from any location to access data and it does not affect the working of database. Distributed database allows us to store one copy of data at different locations. Its advantage is that if a user wants to access data then the nearest site (location) will provide data so it takes less time.

There are multiple sites (computers) in a distributed database so if one site fails then system will not be useless, because other sites can do their job because the same copy of data is installed on every location. You will not find this thing in centralized database.

Any time new nodes (computers) can be added to the network without any difficulty.

Users do not know about the physical storage of data and it is known as distribution transparency, as we know that ideally, a DBMS must not show the details of where each file is stored or we can say that a DBMS should be distribution transparent.

Q.4 Define Client/Server Architecture.

Ans.: The **Client-Server** software architecture model, distinguishes client systems from server systems, which communicate over a computer network. A client-server application is a distributed system that constitutes of both client and server software. A client software or process may initiate a communication session, while the server waits for a requests from a client.[1]

Client/Server describes the relationship between two computer programs in which one program, the client, makes a service request from another program, the server, which fulfills the request. Although the client/server idea can be used by programs within a single computer, it is a more important idea in a network. In a network, the client/server model provides a convenient way to efficiently interconnect programs that are distributed across different locations. Computer transactions using the client/server model are very common. Most Internet applications, such as email, web access and database access, are based on the client/server model. For example, a web browser is a client program at the user computer that may access information at any web server in the world. To check your bank account from your computer, a web browser client program in your computer forwards your request to a web server program at the bank. That program may in turn forward the request to its own database client program that sends a request to a database server at another bank computer to retrieve your account balance. The balance is returned back to the bank database client, which in turn serves it back to the web browser client in your personal computer, which displays the information for you.

The Client/Server model has become one of the central ideas of network computing. Most business applications being written today use the client/server model. So does the Internet's main application protocols, such as HTTP, SMTP, Telnet, DNS, etc. In marketing, the term has been used to distinguish distributed computing by smaller dispersed computers from the "monolithic" centralized computing of mainframe characteristics computers. But this distinction has largely disappeared as mainframes and their applications have also turned to the client/server model and become part of network computing.

Each instance of the client software can send data requests to one or more connected *server*s. In turn, the servers can accept these requests, process them, and return the requested information to the client. Although this concept can

be applied for a variety of reasons to many different kinds of applications, the architecture remains fundamentally the same.

The most basic type of client-server architecture employs only two types of hosts: clients and servers. This type of architecture is sometimes referred to as *two-tier*. It allows devices to share files and resources.

These days, clients are most often web browsers, although that has not always been the case. Servers typically include web servers, database servers and mail servers. Online gaming is usually client-server too. In the specific case of MMORPG, the servers are typically operated by the company selling the game; for other games one of the players will act as the host by setting his game in server mode.

The interaction between client and server is often described using sequence diagrams. Sequence diagrams are standardized in the Unified Modeling Language.

When both the client- and server-software are running on the same computer, this is called a *single seat* setup.

Characteristics of a client

Request sender is known as Client :

Initiates requests.

Waits for and receives replies.

Usually connects to a small number of servers at one time.

Typically interacts directly with end-users using a graphical user interface.

Characteristics of a Server :

Receiver of request which is sent by client is known as server.

Passive (slave).

Waits for requests from clients.

Upon receipt of requests, processes them and then serves replies.

Usually accepts connections from a large number of clients.

Typically does not interact directly with end-users.

□ □ □

Chapter-7

Data and Query Processing

Q.1 Discuss the factors which determines the role of Data Processing?

Ans.: It is important to understand information system in the context of their use in information processing. Which is also called data processing. We can define data or information processing as the processing of data to make it more usable and meaningful, Thus transforming it into information. The factors which determines the role of data processing are as follows -

Data versus Information :

	Data	Information
Meaning:	Plain facts	When data are processed, organized, structured or presented in a given context so as to make them useful, they are called Information.

Data are plain facts. The word "data" is plural for "datum." When data are processed, organized, structured or presented in a given context so as to make them useful, they are called **Information**.

It is not enough to have data (such as statistics on the economy). Data themselves are fairly useless. But when these data are *interpreted* and processed to determine its true *meaning*, they becomes useful and can be called Information.

The MIS versus The Data Processing System :

Data Processing Systems, or **DP Systems**, are concerned with transaction handling and record-keeping, usually for a particular functional area.

Here are a few differences between an MIS and a DPS :

The integrated database of an MIS enables greater flexibility in meeting the information needs of management.

An MIS integrates the information flow between functional areas (accounting, marketing, inventory management, etc.), whereas DP systems tend to support a single functional area.

An MIS caters to the information needs of all levels of management, whereas DP systems focus on the clerical and operational levels.

Management's information needs are supported on a timelier basis with an MIS than they are with a DP system. An MIS, for example, has online inquiry capability for the immediate generation of reports, whereas a DP system usually produces only scheduled reports.

Q.2 How will you define Database Query Language?

Ans.: Query Languages are computer languages used to make queries into databases and information systems.

Broadly, query languages can be classified according to whether they are database query languages or information retrieval query languages. Examples include:

QL is a proprietary object-oriented query language for querying relational databases.

Common Query Language (CQL) a formal language for representing queries to information retrieval systems such as web indexes or bibliographic catalogues.

CODASYL D is a query language for truly relational database management systems (TRDBMS);

DMX is a query language for Data Mining models;

Datalog is a query language for deductive databases;

ERROL is a query language over the Entity-Relationship Model (ERM), especially tailored for relational databases;

Gellish English is a language that can be used for queries in Gellish English Databases [1], for dialogues (requests and responses) as well as for information modeling and knowledge modeling;

ISBL is a query language for PRTV, one of the earliest relational database management systems;

LDAP is an application protocol for querying and modifying directory services running over TCP/IP.

MQL is a cheminformatics query language for a substructure search allowing beside nominal properties also numerical properties;

MDX is a query language for OLAP databases;

OQL is Object Query Language;

OCL (Object Constraint Language). Despite it's name, OCL is also an object query language and a OMG standard.

OPath, intended for use in querying WinFS *Stores*;

Poliqarp Query Language is a special query language designed to analyze annotated text. Used in the Poliqarp search engine;

QUEL is a relational database access language, similar in most ways to SQL;

SMARTS is the cheminformatics standard for a substructure search;

SPARQL is a query language for RDF graphs;

SQL is a well known query language for relational databases;

SuprTool is a proprietary query language for SuprTool, a database access program used for accessing data in Image/SQL (TurboIMAGE) and Oracle databases;

TMQL Topic Map Query Language is a query language for Topic Maps;

XQuery is a query language for XML data sources;

Q.3 What is Query Processing?

Ans.: Query Processing is the procedure of selecting the best plan or strategies to be used in responding to a database request. Query is actually a stepwise process ,in which first step is to transform thequery into standard form. For example, Query is QBE is translated into SQL, and subsequently into a relation algebric expression during the tranformation parser perform portion of the query processor checks the syntax and varifies.

In the next step a number of strategies called access plans are generated for evaluating the tranformed query.the cost of each access plan is estimeted and the optimal one is chosen and executed.

STUDENT (std# ,std-name)

REGISTRATION(std#,course#)

GRADE(std#,course#,grade)

COURSE (course#,course-name,instructor)

Genral strategies for Query processing:

Query Representation :

Query posed by users are not in a form which is convenient for internal system use .So query procesdsors tranform it into thre form of relational calculus ,relational algebra, Object graph,operator graph or tables.

Operator Graph :

An operator graph despicts how a sequence of operation can be performed. In operator graph s, operations are represented by nodes and the flow of data is shown by directed edges.

Example Query : ¯List names students registered in database course‖

Πstd-name(σcourse-name=‖database‘(STUDENT |X| REGISTRATION|X| COURSE)

Steps in Query Processing :

Convert to a Standard Strate Point : We can use relational algebric form and operator graph as the starting point.

We can assume that the query expression is in conjunctive normal form.

Transform the Query : The query is transformed by replasing expression in the query with those that are likely to enhance performance.

Simplify the Query : The query is simplified by removing redundant and useless operations.

Prepare Alternative Access Plans : The alternative access plan indicates the order in which the various operations will be performed and the cost of such plan.The cost is depends upon whether or the not the relations. Are sorted and the presence or absence of indexes. The optional aceess plan is chosen.

Genral Processing Straitegies :

Perform selection as early as possible.

Combine a number of unary operations.

Convert the cartesian product with a certain subsequent selection into join.

Compute common expression once.

Preprocess the relations.

Query Evaluation Plans :

We can classify the query evaluation approaches according to the number of relations involved in query expression.

These are as follows :

One Variable Expression

Sequential Access

Two Variable Expression

Nested Loop Method

Sort and Merge Method

Hash Join.

□□□

Chapter-8

Structured Query Language

Q.1 Describe basic SQL with its Procedural Extensions.

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

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

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

Originally designed as a declarative query and data manipulation language, variations of SQL have been created by SQL database management system (DBMS) vendors that add procedural constructs, control-of-flow statements, user-defined data types, and various other language extensions. With the release of the SQL:1999 standard, many such extensions were formally adopted as part of the SQL language via the SQL Persistent Stored Modules (SQL/PSM) portion of the standard.Common criticisms of SQL include a perceived lack of cross-platform portability between vendors, inappropriate handling of missing data, and unnecessarily complex and occasionally ambiguous language grammar and semantics.

History :

During the 1970s, a group at ***IBM's*** San Jose research center developed the System R relational database management system, based on the model introduced by Edgar F. Codd in his influential paper, **A Relational Model of Data for Large Shared Data Banks**.[3] Donald D. Chamberlin and Raymond F. Boyce of IBM subsequently created the **Structured English Query Language** (SEQUEL) to manipulate and manage data stored in System R.[4] The acronym SEQUEL was later changed to SQL because "SEQUEL" was a trademark of the UK-based Hawker Siddeley aircraft company.

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

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

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

between vendors, inappropriate handling of missing data, and unnecessarily complex and occasionally ambiguous language grammar and semantics.

SQL	
Paradigm	Multi-paradigm
Appeared in	1974
Designed by	Donald D. Chamberlin and Raymond F. Boyce
Developer	IBM

Latest release	SQL:2006/ 2006
Typing discipline	static, strong
Major implementations	Many
Dialects	SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006
Influenced by	Datalog
Influenced	CQL, LINQ, Windows PowerShell
OS	Cross-platform

Standardization :

SQL was adopted as a standard by ANSI in 1986 and ISO in 1987. In the original SQL standard, ANSI declared that the official pronunciation for SQL is "es queue el". However, many English-speaking database professionals still use the nonstandard pronunciation /sikwel/ (like the word "sequel"). Until 1996, the National Institute of Standards and Technology (NIST) data management standards program was tasked with certifying SQL DBMS compliance with the SQL standard. In 1996, however, the NIST data management standards program was dissolved, and vendors are now relied upon to self-certify their products for compliance.

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

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First published by ANSI. Ratified by ISO in 1987.
1989	SQL-89	FIPS 127-1	Minor revision, adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2.

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

The SQL standard is not freely available. SQL:2003 and SQL:2006 may be purchased from ISO or ANSI. A late draft of SQL:2003 is freely available as a

zip archive, however, from Whitemarsh Information Systems Corporation. The zip archive contains a number of PDF files that define the parts of the SQL:2003 specification. Scope and extensions.

Procedural Extensions :

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

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

In addition to the standard SQL/PSM extensions and proprietary SQL extensions, procedural and object-oriented programmability is available on many SQL platforms via DBMS integration with other languages. The SQL standard defines SQL/JRT extensions (SQL Routines and Types for the Java Programming Language) to support Java code in SQL databases. SQL Server 2005 uses the SQLCLR (SQL Server Common Language Runtime) to host managed .NET assemblies in the database, while prior versions of SQL Server

were restricted to using unmanaged extended stored procedures which were primarily written in C. Other database platforms, like MySQL and Postgres, allow functions to be written in a wide variety of languages including Perl, Python, Tcl, and C.

Q.2 Define Keys.

Ans.: i) **Primary Key :** Most DBMSs require a table to be defined as having a single key, rather than a number of possible keys. A primary key is a key which the database designer has designated for this purpose any record can be identified by the use of primary key. Primary key should not be null. There are basically two constraints for being a primary key :

It should not be null.

Itshould not be repeated.

Super Key : A superkey is an attribute or set of attributes that uniquely identifies rows within a table; in other words, two distinct rows are always guaranteed to have distinct superkeys. {Employee ID, Employee Address, Skill} would be a superkey for the "Employees' Skills" table; {Employee ID, Skill} would also be a superkey.

Candidate Key : A candidate key is a minimal superkey, that is, a superkey for which we can say that no proper subset of it is also a superkey. {Employee Id, Skill} would be a candidate key for the "Employees' Skills" table. In otherwords, It is a key which is not a primary key itself but having all the properties for being a primary key.

Foreign Key : In the context of relational databases, a Foreign Key is a referential constraint between two tables.[1] The foreign key identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table. The columns in the referencing table must be the primary key or other candidate key in the referenced table. The values in one row of the referencing columns must occur in a single row in the referenced table. Thus, a row in the referencing table cannot contain values that don't exist in the referenced table (except potentially NULL). This way references can be made to link information together and it is an essential part of database normalization. Multiple rows in the referencing table may refer to the same row in the referenced table. Most of the time, it reflects the one (master table, or referenced table) to many (child table, or referencing table) relationship.

Unique Key : A unique key is the key by which a user can identify a domain uniquely. It is different from the primary key in context that a unique key can be null.

Q.3 Describe the Language Elements of SQL.

Ans.: Language Elements :

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

Statements which may have a persistent effect on schemas and data, or which may control transactions, program flow, connections, sessions, or diagnostics.

Queries which retrieve data based on specific criteria.

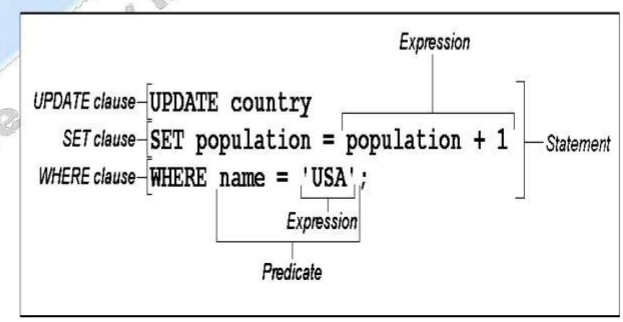
Expressions which can produce either scalar values or tables consisting of columns and rows of data.

Predicates which specify conditions that can be evaluated to SQL three-valued logic (3VL) Boolean truth values and which are used to limit the effects of statements and queries, or to change program flow.

Clauses which are (in some cases optional) constituent components of statements and queries.

Whitespace is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.



Queries :

The most common operation in SQL databases is the query, which is performed with the declarative SELECT keyword. SELECT retrieves data from a specified table, or multiple related tables, in a database. While often grouped with Data Manipulation Language (DML) statements, the standard

SELECT query is considered separate from SQL DML, as it has no persistent effects on the data stored in a database. Note that there are some platform-specific variations of SELECT that can persist their effects in a database, such as the SELECT INTO syntax that exists in some databases.

SQL queries allow the user to specify a description of the desired result set, but it is left to the devices of the database management system (DBMS) to plan, optimize, and perform the physical operations necessary to produce that result set in as efficient a manner as possible. An SQL query includes a list of columns to be included in the final result immediately following the SELECT keyword. An asterisk ("*") can also be used as a "wildcard" indicator to specify that all available columns of a table (or multiple tables) are to be returned. SELECT is the most complex statement in SQL, with several optional keywords and clauses.

The following is an example of a SELECT query that returns a list of expensive books. The query retrieves all rows from the *books* table in which the *price* column contains a value greater than 100.00. The result is sorted in ascending order by *title*. The asterisk (*) in the *select list* indicates that all columns of the *books* table should be included in the result set.

SELECT *

FROM books

WHERE price > 100.00

ORDER BY title;

The example below demonstrates the use of multiple tables in a join, grouping, and aggregation in an SQL query, by returning a list of books and the number of authors associated with each book.

SELECT books.title, count(*) **AS** Authors

FROM books

JOIN book_authors

ON books.isbn = book_authors.isbn

GROUP BY books.title;

Example output might resemble the following :

Title Authors

---------- --------

SQL Examples and Guide 3

The Joy of SQL 1

How to use Wikipedia 2

Pitfalls of SQL 1

How SQL Saved my Dog 1

(The underscore character "_" is often used as part of table and column names to separate descriptive words because other punctuation tends to conflict with SQL syntax. For example, a dash "-" would be interpreted as a minus sign.)

Under the precondition that *isbn* is the only common column name of the two tables and that a column named *title* only exists in the *books* table, the above query could be rewritten in the following form:

SELECT title, count(*) **AS** Authors

FROM books

NATURAL JOIN book_authors

GROUP BY title;

However, many vendors either don't support this approach, or it requires certain column naming conventions. Thus, it is less common in practice.

Data retrieval is very often combined with data projection when the user is looking for calculated values and not just the verbatim data stored in primitive data types, or when the data needs to be expressed in a form that is different from how it's stored. SQL allows the use of expressions in the *select list* to project data, as in the following example which returns a list of books that cost more than 100.00 with an additional *sales_tax* column containing a sales tax figure calculated at 6% of the *price*.

SELECT isbn, title, price, price * 0.06 **AS** sales_tax

FROM books

WHERE price > 100.00

ORDER BY title;

Data manipulation :

First, there are the standard Data Manipulation Language (DML) elements. DML is the subset of the language used to add, update and delete data:

INSERT is used to add rows (formally tuples) to an existing table, for example :

INSERT INTO my_table (field1, field2, field3) **VALUES** ('test', 'N', **NULL**);

UPDATE is used to modify the values of a set of existing table rows, eg:

UPDATE my_table **SET** field1 = 'updated value' **WHERE** field2 = 'N';

DELETE removes zero or more existing rows from a table, eg:

DELETE FROM my_table **WHERE** field2 = 'N';

MERGE is used to combine the data of multiple tables. It is something of a combination of the INSERT and UPDATE elements. It is defined in the SQL:2003 standard; prior to that, some databases provided similar functionality via different syntax, sometimes called an "upsert".

Transaction controls :

Transactions, if available, can be used to wrap around the DML operations:

START TRANSACTION (or BEGIN WORK, or BEGIN TRANSACTION, depending on SQL dialect) can be used to mark the start of a database transaction, which either completes completely or not at all.

COMMIT causes all data changes in a transaction to be made permanent.

ROLLBACK causes all data changes since the last COMMIT or ROLLBACK to be discarded, so that the state of the data is "rolled back" to the way it was prior to those changes being requested.

Once the COMMIT statement has been executed, the changes *cannot* be rolled back. In other words, its meaningless to have ROLLBACK executed after COMMIT statement and vice versa.

COMMIT and ROLLBACK interact with areas such as transaction control and locking. Strictly, both terminate any open transaction and release any locks held on data. In the absence of a START TRANSACTION or similar statement, the semantics of SQL are implementation-dependent. Example: *A classic bank transfer of funds transaction.*

```
START TRANSACTION;
UPDATE ACCOUNTS SET AMOUNT=AMOUNT-200 WHERE
ACCOUNT_NUMBER=1234;
UPDATE ACCOUNTS SET AMOUNT=AMOUNT+200 WHERE
ACCOUNT_NUMBER=2345;
IF ERRORS=0 COMMIT;
IF ERRORS<>0 ROLLBACK;
```

Data Definition :

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

CREATE causes an object (a table, for example) to be created within the database.

DROP causes an existing object within the database to be deleted, usually irretrievably.

TRUNCATE deletes all data from a table (non-standard, but common SQL statement).

ALTER statement permits the user to modify an existing object in various ways -- for example, adding a column to an existing table.

Example :

```
CREATE TABLE my_table (
  my_field1  INT,
  my_field2 VARCHAR (50),
  my_field3  DATE       NOT NULL,
  PRIMARY KEY (my_field1, my_field2)
);
```

Data Control :

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

GRANT authorizes one or more users to perform an operation or a set of operations on an object.

REVOKE removes or restricts the capability of a user to perform an operation or a set of operations.

Example :

```
GRANT SELECT, UPDATE ON my_table TO some_user, another_user
```

Other :

ANSI-standard SQL supports double dash, --, as a single line comment identifier (some extensions also support curly brackets or C style /* comments */ for multi-line comments).

Example :

```
SELECT * FROM inventory -- Retrieve everything from inventory table
```

Some SQL servers allow user-defined functions Criticisms of SQL

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

In addition, there are also some criticisms about the practical use of SQL:

Implementations are inconsistent and, usually, incompatible between vendors. In particular date and time syntax, string concatenation, nulls, and comparison case sensitivity often vary from vendor to vendor.

The language makes it too easy to do a Cartesian join (joining all possible combinations), which results in "run-away" result sets when WHERE clauses are mistyped. Cartesian joins are so rarely used in practice that requiring an explicit CARTESIAN keyword may be warranted. *SQL 1992* introduced the CROSS JOIN keyword that allows the user to make clear that a cartesian join is intended, but the shorthand "comma-join" with no predicate is still acceptable syntax.

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

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

Reasons for Lack of Portability :

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

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

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

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

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

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

Q.4 What are the alternatives of SQL?

Ans.: Alternatives to SQL :

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

IBM Business System 12 (IBM BS12)

Hibernate Query Language (HQL) - A Java-based tool that uses modified SQL

Quel introduced in 1974 by the U.C. Berkeley Ingres project.

Object Query Language

QL - object-oriented Datalog and QBE (Query By Example) create by Moshe Zloof

LINQ

4D Query Language (4D QL) and so many other languages which can be used as the alternatives of SQL.

Q.5 Define QBEL including Aggregate Operations.

Ans.: QBE is both a query language and the name of a DB system including it. The system is no longer in use, but the language is part of IBM's Query Management Facility (QMF).

Basic Structure :

QBE has ``two-dimensional'' syntax.

Queries are expressed by example.

Close correspondence with domain relational calculus.

Non-procedural.

Queries are expressed using **skeleton tables**.

User selects the skeletons needed.

User fills in skeletons with example rows.

An example row consists of constants and example elements which are really domain variables.

Domain variables are preceded by an underscore character.

Constants appear without any qualification.

Simple Queries :

For example, to find all customers having an account at the SFU branch :

deposit	bname	account#	cname	balance
	SFU		P._x	

deposit	bname	account#	cname	balance
	SFU		P._x	

A P. before the variable causes printing.

A P.ALL. prefix suppresses duplicate elimination.

A P. in front of the row prints all attributes.

The domain variable may be omitted if it is not used elsewhere.

Arithmetic expressions are allowed.

Comparison operators are allowed, space on left hand side is left blank.

To find the names of all branches not located in Burnaby :

branch	bname	assets	bcity
	P.		¬ Burnaby

branch	bname	assets	bcity
	P.		¬ Burnaby

To find all customers having an account at both the SFU and the MetroTown branch :

deposit	*bname*	*account#*	*cname*	*balance*
	SFU		P._x	
	MetroTown		_x	

deposit	bname	account#	cname	balance
	SFU MetroTown		P._x _x	

To find all customers having an account at either branch or both :

deposit	*bname*	*account#*	*cname*	*balance*
	SFU		P._x	
	MetroTown		P._y	

deposit	bname	account#	cname	balance
	SFU MetroTown		P._x P._y	

Find all customers having an account at the same branch as John :

deposit	*bname*	*account#*	*cname*	*balance*
	_x		Jones	
	_x		P._y	

deposit	bname	account#	cname	balance
	_x _x		John P._y	

Queries on Several Relations :

Queries on several relations require several skeleton tables.

To find the name and city of all customers having a loan at the SFU branch :

borrow	bname	loan#	cname	amount
	SFU		_x	

borrow	bname	loan#	cname	amount
	SFU		_x	

customer	cname	street	ccity
	P._x		P._y

customer	cname	street	ccity
	P._x		P._y

Find the name of all customers having an account at the SFU branch, but no loan from that branch.

Queries involving negation can be expressed by putting a sign under the relation name beside an example row :

deposit	bname	account#	cname	balance
	SFU		P._x	

deposit	bname	account#	cname	balance
	SFU		P._x	

borrow	bname	loan#	cname	amount
¬	SFU		_x	

deposit	bname	loan#	cname	amount
¬	SFU		_x	

To find all customers who have accounts at two different branches : -

deposit	bname	account#	cname	balance
	_y		P._x	
	¬ _y		_x	

deposit	bnan	eaccount#	cnar	ebalance
	_y ¬ _y		P._x _x	

The Condition Box :

When it is difficult or impossible to express all constraints on the domain variables within the skeleton tables, the **condition box** may be used.

To add the constraint that we are only interested in customers other than John to the above query, we include the condition box:

conditions
_x ¬= Jones

conditions
_x ¬=John

To find all account numbers with balances between $1,300 and $1,500 :

deposit	*bname*	*account#*	*cname*	*balance*
		P.		_x

deposit	bname	account#	cname	balance
		P.		_x

conditions
_x ≥ 1300
_x ≤ 1500

conditions
_x ≥1300
_x ≤1500

Logical expressions **and** and **or** may appear in the condition box.

To find all account numbers where the balance is between $1,300 and $2,000, but is not $1,500 :

deposit	*bname*	*account#*	*cname*	*balance*
		P.		_x

deposit	bname	account#	cname	balance
		P.		_x

conditions
_x = (≥ 1300 **and** ≤ 2000 **and** ¬ 1500)

conditions
_x =(1300 and ≤ 2000 and ¬ 1500

An unconventional use of the or construct allows comparison with several constant values :

conditions
_x = (Burnaby **or** Richmond)

conditions
_x = (Burnaby or Richmond)

The Result Relation :

If the result of a query includes attributes from several relation schemes, we need a way of displaying the result in a single table.

We can declare a temporary *result* relation including the attributes to be displayed. We put the print command only in that table.

To find the customer names and cities and account numbers for all customers having an account at the SFU branch :

deposit	*bname*	*account#*	*cname*	*balance*
	SFU	_z	_x	

deposit	bname	account#	cname	balance
	SFU	_z	_x	

customer	*cname*	*street*	*ccity*
	_x		_y

customer	cname	street	ccity
	_x		_y

result	*cname*	*ccity*	*account#*
P.	_x	_y	_z

result	cname	ccity	account #
P.	_x	_y	_z

Ordering the Display of Tuples :

The order in which tuples are displayed can be controlled by adding the command AO. (ascending order) or DO. (descending order) to the print command :

deposit	*bname*	*account#*	*cname*	*balance*
	SFU		P.AO.	

deposit	bname	account#	cname	balance
	SFU		P.A.O.	

To sort first by name, and then by balance for those with multiple accounts :

deposit	bname	account#	cname	balance
	SFU		P.AO(1).	P.DO(2).

deposit	bname	account#	cname	balance
	SFU		P.AO (1)	P.DO (2)

Aggregate Operations :

QBE includes the aggregate operators AVG, MAX, MIN, SUM and CNT. As QBE eliminates duplicates by default, they must have ALL. appended to them.

To find the total balance of all accounts belonging to John :

deposit	bname	account#	cname	balance
			Jones	P.SUM.ALL.

deposit	bname	account#	cname	balance
			John	P.SUM.ALL.

All aggregate operators must have ALL. appended, so to override the ALL.\ we must add UNQ. (unique). (NOTE: a number of examples in the text incorrectly show UNQ. replacing ALL.)

deposit	bname	account#	cname	balance
	Main		P.CNT.UNQ.ALL.	

deposit	bname	account#	cname	balance
	Main		P.CNT.UNQ.ALL.	

To compute functions on groups, we use the G. operator. To find the average balance at each branch :

deposit	bname	account#	cname	balance
	P.G.			P.AVG.ALL._x

deposit	bname	account#	cname	balance
	P.G.			P.AVG.ALL.x

To find the average balances at only branches where the average is more than $1,200, we add the condition box :

conditions
AVG.ALL._x > 1200

conditions
AVG.ALL._x > 1200

To find all customers who have an account at all branches located in Burnaby, we can do :

deposit	*bname*	*account#*	*cname*	*balance*
	_y		P.G._x	

deposit	bname	account#	cname	balance
	_y		P.G._x	

branch	*bname*	*assets*	*bcity*
	_y		Burnaby
	_z		Burnaby

branch	bname	assets	bcity
	_y		Burnaby
	_z		Burnaby

conditions
CNT.UNQ.ALL._y = CNT.UNQ.ALL._z

conditions
CNT.UNQ.ALL._Y= CNT.UNQ.ALL._Z

Modifying the Database :

QBE has facilities for modifying the database.

We simply use D. instead of the P. operator. Whole tuples may be deleted, or only some columns.

Delete all of Smith's account records :

deposit	*bname*	*account#*	*cname*	*balance*
D.			Smith	

deposit	bname	account#	cname	balance
D.			Smith	

Delete the branch-city value for the SFU branch :

branch	*bname*	*assets*	*bcity*
	SFU		D.

branch	bname	assets	bcity
	SFU		D.

Delete all loans with loan numbers between 1300 and 1500 :

borrow	*bname*	*loan#*	*cname*	*amount*
D.		_x		

borrow	bname	loan#	cname	amount
D.		_x		

conditions
$\_x = (\geq 1300$ **and** $\leq 1500)$

conditions
_x =(1300 and ≤ 1500)

Delete all accounts at branches located in Burnaby :

deposit	*bname*	*account#*	*cname*	*balance*
D.	_x			

deposit	bname	account#	cname	balance
D.	_x			

branch	*bname*	*assets*	*bcity*
	_x		Burnaby

branch	bname	assets	bcity
	_x		Burnaby

Insertion :

Insertion uses the I. operator.

To insert an account tuple for Smith :

deposit	*bname*	*account#*	*cname*	*balance*
I.	SFU	9372	Smith	1200

deposit	bname	account#	cname	balance
1.	SFU	9372	Smith	1200

If values are missing, **nulls** are inserted.

To provide all loan customers in the SFU branch with a $200 savings account :

deposit	*bname*	*account#*	*cname*	*balance*
I.	SFU	_x	_y	200

deposit	bname	account#	cname	balance
1.	SFU	_x	_y	200

borrow	*bname*	*loan#*	*cname*	*amount*
	SFU	_x	_y	

borrow	bname	loan#	cname	amount
	SFU	_x	_y	

Updates :

We can update individual attributes with the U. operator. Fields left blank are not changed.

To update the assets of the SFU branch to $10,000,000 :

branch	*bname*	*assets*	*bcity*
	SFU	U.10000000	

branch	bname	assets	bcity
	SFU	U.10000000	

To make interest payments of 5% on all balances :

deposit	*bname*	*account#*	*cname*	*balance*
U.				_x * 1.05
				_x

Deposit	bname	account#	cname	balance
U.				_x*1.05 _x

Q.6 Define Quel with its capabilities.

Ans.: Quel :

We will not cover this section, aside from making a few remarks about this language.

Quel was the original query language for the **Ingres** dbms. Ingres is now available with SQL.

Quel closely resembles the tuple relational calculus.

Queries use the **range of, retrieve** and **where** clauses.

A typical query : **range of** *t* **is**

borrow **range of** *s* **is** *deposit*

retrieve unique (*s.cname*)

where *t.bname*=``SFU''

and *s.bname*=``SFU'' **and**

t.cname=s.cname

This finds the names of all customers who have both a loan and an account at the SFU branch.

There is no representation for or in Quel.

Quel has the power of the relational algebra by means of the **any** aggregate function and the use of insertion and deletion into temporary relations.

Q.7 How can you convert any query easily into any other language?

Ans.: Converting Queries Easily into Any Language :

Decide on the relations required to answer the query.

You'll need relations containing attributes explicitly mentioned, plus relations needed to ``traverse'' between needed relations.

In some cases you will need more than one copy of a relation.

Don't include unneeded relations.

Draw them on a piece of paper.

It helps to draw them in a sensible order.

Draw them in the order you would ``traverse'' them. This will simplify the drawing of links.

Draw in links and constant values.

Put links between attributes in different relations wherever the attributes are required to satisfy some comparison operator (*equals, less than*, etc.).

Ordinary lines are used for *equals*, and write any other comparison operator on the line at some convenient spot.

Write in constant values, where some attribute must have a specific value.

For simpler queries, the following advice works. Where you need set operations or division, a little more thought is needed.

Relational Algebra : we'll do a correct but not necessarily optimal query.

Do an appropriate combination of **Cartesian products** and **natural joins** of the relations required.

Do a **select** where the predicate demands that all the links and constants in your diagram be true.

Don't forget that natural joins will take care of some of your diagram's links.

Finally, do a **project** of the attributes to be printed out.

Tuple Relational Calculus :

Create a tuple variable for each of the relations in your diagram.

Make sure the parentheses give you the required scope.

Ensure each link and constant in your diagram corresponds to some part of your predicate.

Make sure *t* gets the attributes that should be printed out.

Domain Relational Calculus :

Create domain variables. Name them sensibly.

Remember that equality is forced by using the same domain variable in several places.

Other comparison operators may be explicitly stated, e.g.

Remember to use the existential qualifier for domain variables, and to make sure your scoping is correct.

SQL : Similar to relational algebra.

Put all the relations needed in the **from** clause.

Remember to use **tuple variables** when you have more than one copy of a relation, or for general convenience.

Express each of the links and constants in your diagram as part of the predicate in the **where** clause.

State the attributes to be printed out in the **select** clause.

QBE : Your diagram is almost QBE to start with.

Select the skeleton tables needed.

Remember that you only need **one** skeleton table per relation. You can put more than one line in a skeleton table.

Force equality on links by using the same domain variables in different places (see the connection to domain relational calculus?).

Use the **condition box** where necessary.

Use **P.** to print out the attributes. Remember to use a **result relation** if attributes are printed out from more than one skeleton table.

□ □

Chapter-9

Advanced Features of SQL

Q.1 What is Embedded SQL? Explain in detail.

Ans.: Embedded SQL is a method of combining the computing power of a programming language and the database manipulation capabilities of SQL. It allows programmers to embed SQL statements in programs written in C/C++, Fortran, COBOL, Pascal, etc.

Embedded SQL statements are SQL statements written within application programming languages and preprocessed by a SQL preprocessor before the application program is compiled. There are two types of embedded SQL: static and dynamic.

The SQL standard defines embedding of SQL as embedded SQL and the language in which SQL queries are embedded is referred to as the host language. A popular host language is C. The mixed C and embedded SQL is called Pro*C in Oracle and Sybase database management systems. Other embedded SQL precompilers are Pro*COBOL, Pro*FORTRAN, Pro*PL/I, Pro*Pascal, and SQL*Module (for Ada).

Embedded SQL is a superset of Sybase's T-SQL or Oracle's PL/SQL that lets you place SQL statements in application programs written in languages such as C and COBOL. Pro*C allows the C programmer to write database access code fast and with less of a learning curve. For people who are familiar with both C and SQL, this is a cakewalk. Its worth noting that there are differences between implementations of Pro*C across different database vendors due to the differences between database architectures, datatypes, etc. Each new release of a database may announce certain enhancements or changes to its Embedded SQL pre-compiler. It is best to track changes on this front by referring to the vendor database websites.

A Pro*C program is compiled in two steps. First, the Pro*C precompiler recognizes the SQL statements embedded in the program, and replaces them with appropriate calls to the functions in the SQL runtime library. The output

is pure C/C++ code with all the pure C/C++ portions intact. Then, a regular C/C++ compiler is used to compile the code and produces the executable.

Pro*C Syntax

SQL

All SQL statements need to start with EXEC SQL and end with a semicolon ";". You can place the SQL statements anywhere within a C/C++ block, with the restriction that the declarative statements do not come after the executable statements. As an example :

```
{
   int a;
   /* ... */
   EXEC SQL SELECT salary INTO :a
         FROM   Employee
         WHERE SSN=876543210;
   /* ... */
   printf("The salary is %d\n", a);
   /* ... */
}
```

Q.2 What is Dynamic SQL? Explain it with suitable example.

Ans.: Dynamic SQL is an enhanced form of Structured Query Language (SQL) that, unlike standard (or static) SQL, facilitates the automatic generation and execution of program statements. This can be helpful when it is necessary to write code that can adjust to varying databases, conditions, or servers. It also makes it easier to automate tasks that are repeated many times.

Dynamic SQL statements are stored as strings of characters that are entered when the program runs. They can be entered by the programmer or generated by the program itself, but unlike static SQL statements, they are not embedded in the source program. Also in contrast to static SQL statements, dynamic SQL statements can change from one execution to the next.

Dynamic SQL statements can be written by people with comparatively little programming experience, because the program does most of the actual

generation of the code. A potential problem is reduced performance (increased processing time) if there is too much dynamic SQL running at any given time.

Q.3 What are the basic requirements for Dynamic SQL Statements?

Ans.: To represent a dynamic SQL statement, a character string must contain the text of a valid SQL statement, but not contain the EXEC SQL clause, or the statement terminator, or any of the following embedded SQL commands :

- CLOSE
- DECLARE
- DESCRIBE
- EXECUTE
- FETCH
- INCLUDE
- OPEN
- PREPARE

Example :

If we need to find all records from the customers table where City = 'London'. This can be done easily such as the following example shows in SQL.

```
DECLARE @city varchar(75)
SET @city = 'London'
SELECT * FROM customers WHERE City = @city
```

In Dynamic SQL we can write it as

```
DECLARE @sqlCommand varchar(1000)

DECLARE @columnList varchar(75)

DECLARE @city varchar(75)

SET @columnList = 'CustomerID, ContactName, City'

SET @city = '''London'''
```

```
SET @sqlCommand = 'SELECT ' + @columnList + ' FROM customers
WHERE City = ' + @city
EXEC (@sqlCommand)
```

Q.4 When to use Dynamic SQL?

Ans.: In practice, static SQL will meet nearly all your programming needs. Use dynamic SQL only if you need its open-ended flexibility. Its use is suggested when one of the following items is unknown at pre-compile time :

- Text of the SQL statement (commands, clauses, and so on)
- The number of host variables
- The data types of host variables
- References to database objects such as columns, indexes, sequences, tables, usernames, and views

Q.5 What is Cursor? Define the working of SQL with Cursors.

Ans.: In database packages, the term cursor refers to a control structure for the successive traversal (and potential processing) of records in a result set.

A cursor is used for processing individual rows returned by the database system for a query. It is necessary because many programming languages suffer from impedance mismatch. Programming languages are often procedural and do not offer any mechanism for manipulating whole result sets at once. Therefore, the rows in a result set must be processed sequentially by the application. In this way, a cursor can be thought of as an iterator over the collection of rows in the result set.

Working with Cursors :

A cursor is made known to the DBMS with the DECLARE CURSOR statement. A name has to be assigned for the cursor.

```
DECLARE cursor_name CURSOR FOR SELECT ... FROM ...
```

Before being used, a cursor must be opened with the OPEN statement. As a result of the opening, the cursor is positioned before the first row in the result set.

OPEN cursor_name

A cursor is positioned on a specific row in the result set with the FETCH statement. A fetch operation transfers the data of the row into the application. Once all rows are processed or the fetch operation is to be positioned on a non-existing row (cf. scrollable cursors below), a SQLSTATE '02000' (usually accompanied by an SQLCODE +100) is returned by the DBMS to indicate the end of the result set.

FETCH cursor_name INTO ...

The last step is to close the cursor using the CLOSE statement.

CLOSE cursor_name

Once a cursor is closed it can be opened again, which implies that the query is evaluated again and a new result set is built.

□ □ □

Chapter-10

Query Optimization Techniques

Q.1 What do you mean by Query Optimization? What are the Query Evaluation Plans in RDBMS? How will you define Relational Query Optimization?

Ans.: Query optimization is a function of many relational database management systems in which multiple query plans for satisfying a query are examined and a good query plan is identified. This may or not be the absolute best strategy because there are many ways of doing plans. There is a trade off between the amount of time spent figuring out the best plan and the amount running the plan. Different qualities of database management systems have different ways of balancing these two. Cost based query optimizers evaluate the resource footprint of various query plans and use this as the basis for plan selection. Typically the resources which are costed as CPU path length, amount of disk buffer space, disk storage service time, and interconnect usage between units of parallelism. The set of query plans examined is formed by examining possible access paths (e.g., primary index access, secondary index access, full file scan) and various relational table join techniques (e.g, merge join, hash join, product join). The search space can become quite large depending on the complexity of the SQL query. There are two types of optimization. These consist of logical optimization which generates a sequence of relational algebra to solve the query. In addition there is physical optimization which is used to determine the means of carrying out each operation.

Translating SQL Queries into Relational Algebra : Different optimizer sequences such as the different Greek alphabet characters which are assigned to them. This is explained in the article relational algebra.

The Goal of Query Optimization : The goal is to eliminate as many unneeded tuples, or rows as possible. The following is a look at relational algebra as it eliminates unneeded tuples. The project operator is straightforward to implement if <attribute list> contains a key to relation R. If

it does not include a key of R, it must be eliminated. This must be done by sorting (see sort methods below) and eliminating duplicates. This method can also use hashing to eliminate duplicates Hash table.

Relational Query Optimizer : The query optimizer is the heart of RDBMS performance and must also be extended with knowledge about how to execute User Defined Functions efficiently, take advantage of new index structures, transform queries in new ways, and navigate among data using references. Successfully opening up such a critical and highly tuned DBMS component and educating third parties about optimization techniques is a major challenge for DBMS vendors.

Q.2 Define Iterator Interface in terms of Relational Operators.

Ans.: Relational Operators and the Iterator Interface : One of the most powerful features of relational databases is their support for declarative query languages such as SQL. With these languages, users describe the output of their queries without needing to specify how that query is supposed to be executed by the DBMS. It is the responsibility of the Query Optimizer to built a query evaluation plan, a tree of relational operators, so that the user's query can be answered with the least incurred cost. Resulttuples flow from the nodes at the lower levels of the tree to their parent nodes, where they get further processed and outputted to their parents respectively, until they reach the top-most node, the root of the operator tree, at which point they are returned to the user.

To simplify the interaction between the different operators in a query plan tree, every operator is required to implement a uniform iterator interface, so that parent nodes in a query plan do not need to explicitly know the type of the operator(s) they accept as input, and thus what entry point functions they need to call to interact with them. The abstract iterator class which all operators of Minibase extend is relop.Iterator. The functions that it provides are the following:

Constructor(Iterator[] inputs, params): Sets up its member attributes (e.g. its iterator inputs { one if it is a unary operator such as selection or projection, two if it is a binary operator like join) and initializes (\opens") the iterator.

isOpen(): Checks whether the iterator is open.

close(): Closes the iterator, and releases any temporary resources (such as temporary ¯les) held within its lifetime.

restart(): Restarts the iterator, i.e. as if it were just constructed.

explain(): A recursive function that gives a one- line explanation of the iterator, and recurses on its inputs

(its children-iterators).

hasNext(): Returns true if the iterator has not visited all the tuples that are to be returned by the operator implementing the interface.

Q.3 How can you explain Pipelined Evaluation in terms of Database Management System.

Ans.: Pipelined Evaluation : As we know, operators in a query plan can be pipelined; as soon as a tuple is generated by an operator which acts as input to another one .In a pipelined plan , Each tuple stream from one operator to another Pipelining allows for parallel execution of operators, avoids unnecessary materialization may be necessary.

(e.g. a selection which might be a left input of a Nested Loops Join (NLJ)),

the higher level operator can immediately include this tuple to its computation. In our example, the NLJ operator can tell whether it has received all the available tuples of its left input by a call to the selection's has Next() method, prior to its next attempt to retrieve the next available tuple. If the selection operator generates for some reason tuples slower than NLJ can consume them, the semantics of hasNext() require the method to block, until a de¯nite answer can be returned (that is whether the selection has exhausted all its input (false), or is still in the process of generating tuples (true)).

It is a good practice to \pre-compute the next available tuple, so that it can be immediately returned by a call to:

getNext(): Retrieves the next available tuple.

□ □ □

Chapter-11

Database Management Issues

Q.1 How will you explain Backup & Recovery for the Database Management System?

Ans.: Backup/Recovery involves the process of making a copy of a database in case of an equipment failure or disaster, then recovering or retrieving the copied database if needed.

One of the innumerable tasks of the DBA is to ensure that all of the databases of the enterprise are always "available." Availability in this context means that the users must be able to access the data stored in the databases, and that the contents of the databases must be up-to-date, consistent, and correct. It must never appear to a user that the system has lost the data or that the data has become inconsistent. This would totally ruin the user's confidence in the database and the entire system.

Many factors threaten the availability of the databases. These include natural disasters (such as floods and earthquakes), hardware failures (for example, a power failure or disk crash), software failures (such as DBMS malfunctions -- read "bugs" -- and application program errors), and people failures (for example, operator errors, user misunderstandings, and keyboard trouble). To this list we can also add the threats such as malicious attempts to destroy or corrupt the contents of the database.

In a large enterprise, the DBA must ensure the availability of several databases, such as the development databases, the databases used for unit and acceptance testing, the operational online production databases (some of which may be replicated or distributed all over the world), the data warehouse databases, the data marts, and all of the other departmental databases. All of these databases usually have different requirements for availability. The online production databases typically must be available, up-to-date, and consistent for 24 hours a day, seven days a week, with minimal

downtime. The warehouse databases must be available and up-to-date during business hours and even for a while after hours.

On the other hand, the test databases need to be available only for testing cycles, but during these periods the testing staff may have extensive requirements for the availability of their test databases. For example, the DBA may have to restore the test databases to a consistent state after each test. The developers often have even more ad hoc requirements for the availability of the development databases, specifically toward the end of a crucial deadline. The business hours of a multinational organization may also have an impact on availability. For example, a working day from 8 a.m. in central Europe to 6 p.m. in California implies that the database must be available for 20 hours a day. The DBA is left with little time to provide for availability, let alone perform other maintenance tasks.

Q.2 Define the basic steps of Recovery Process.

Ans.: Recovery is the corrective process to restore the database to a usable state from an erroneous state. The basic recovery process consists of the following steps -

Identify that the Database is in an Erroneous, Damaged, or Crashed State :

Suspend normal processing.

Determine the source and extent of the damage.

Take corrective action, that is :

Restore the system resources to a usable state.

Rectify the damage done, or remove invalid data.

Restart or continue the interrupted processes, including the re-execution of interrupted transactions.

Resume Normal Processing :

To cope with failures, additional components and algorithms are usually added to the system. Most techniques use recovery data (that is, redundant data), which makes recovery possible. When taking corrective action, the effects of some transactions must be removed, while other transactions must be re-executed; some transactions must even be undone and redone. The recovery data must make it possible to perform these steps.

Q.3 Describe the different techniques which can be used for Recovery from an Erroneous State.

Ans.: The following techniques can be used for recovery from an erroneous state :

Dump and Restart : The entire database must be backed up regularly to archival storage. In the event of a failure, a copy of the database in a previous correct state (such as from a checkpoint) is loaded back into the database. The system is then restarted so that new transactions can proceed. Old transactions can be re-executed if they are available. The following types of restart can be identified :

- A warm restart is the process of starting the system after a controlled system shutdown, in which all active transactions were terminated normally and successfully.
- An emergency restart is invoked by a restart command issued by the operator. It may include reloading the database contents from archive storage.
- A cold start is when the system is started from scratch, usually when a warm restart is not possible. This may also include reloading the database contents from archive storage. Usually used to recover from physical damage, a cold restart is also used when recovery data was lost.

Undo-Redo Processing (also called Roll-Back and Re-Execute) : By using an audit trail of transactions, all of the effects of recent, partially completed transactions can be undone up to a known correct state. Undoing is achieved by reversing the updating process. By working backwards through the log, all of the records of the transaction in question can be traced, until the begin transaction operations of all of the relevant transactions have been reached. The undo operation must be "idempotent," meaning that failures during undo operations must still result in the correct single intended undo operation taking place. From the known correct state, all of the journaled transactions can then be re-executed to obtain the desired correct resultant database contents. The operations of the transactions that were already executed at a previous stage are obtained from the audit trail. The redo operation must also be idempotent, meaning that failures during redo operations must still result in the correct single intended redo operation taking place. This technique can be used when partially completed processes are aborted.

Roll-Forward Processing (also called Reload and Re-Execute) : All or part of a previous correct state (for example, from a checkpoint) is reloaded; the DBA can then instruct the DBMS to re-execute the recently recorded transactions from the transaction audit trail to obtain a correct state. It is typically used when (part of) the physical media has been damaged.

Restore and Repeat : This is a variation of the previous method, where a previous correct state is restored. The difference is that the transactions are merely reposted from before and/or after images kept in the audit trail. The actual transactions are not re-executed: They are merely reapplied from the audit trail to the actual data table. In other words, the images of the updated rows (the effects of the transactions) are replaced in the data table from the audit trail, but the original transactions are not re-executed as in the previous case.

Q.4 Describe the Maintenance & Performance of DBMS in terms of DBA.

Ans.: The DBA has an extensive set of requirements for the tools and facilities offered by the DBMS. These include facilities to back up an entire database offline, facilities to back up parts of the database selectively, features to take a snapshot of the database at a particular moment, and obviously journaling facilities to roll back or roll forward the transactions applied to the database to a particular identified time. Some of these facilities must be used online -- that is, while the users are busy accessing the database. For each backup mechanism, there must be a corresponding restore mechanism -- these mechanisms should be efficient, because we usually have to restore a lost, corrupt, or damaged database at some critical moment, while the users are waiting anxiously (sometimes highly irritated) and the managers are jumping up and down (often ineffectually)! The backup and restore facilities should be configurable -- we may want to stream the backup data to and from multiple devices in parallel, we may want to add compression and decompression (including using third-party compression tools), we may want to delete old backups automatically off the disk, or we may want to label the tapes according to our own standards. We should also be able to take the backup of a database from one platform and restore it on another -- this step is necessary to cater for non-database-related problems, such as machine and operating system failures. For each facility, we should be able to monitor its progress and receive an acknowledgment that each task has been completed successfully.

Some organizations use so-called "hot standby" techniques to increase the availability of their databases. In a typical hot standby scenario, the operations performed on the operational database are replicated to a standby database. If any problems are encountered on the operational database, the users are switched over and continue working on the standby database until the operational database is restored.

Oracle :

Oracle7 Release 7.3 uses full and partial database backups and a redo log for its database backup and recovery operations. The database backup is an operating system backup of the physical files that constitute the Oracle database. The redo log consists of two or more pre-allocated files, which are used to record all changes made to the database. We can also use the export and import utilities to create a backup of a database. Oracle offers a standby database scheme, with which it maintains a copy of a primary database on duplicate hardware, in a constant recoverable state, by applying the redo logs archived off the primary database.

A full backup is an operating system backup of all of the data files, parameter files, and the control file that constitute the database. A full database backup can be taken by using the operating system's commands or by using the host command of the Server Manager. A full database backup can be taken online when the database is open, but only an offline database backup (taken when the database server is shut down) will necessarily be consistent. An inconsistent database backup must be recovered with the online and archived redo log files before the database will become available. The best approach is to take a full database backup after the database has been shut down with normal or immediate priority.

A partial backup is any operating system backup of a part of the full backup, such as selected data files, the control file only, or the data files in a specified tablespace only. A partial backup is useful if the database is operated in ARCHIVELOG mode. A database operating in NOARCHIVE mode rarely has sufficient information to use a partial backup to restore the database to a consistent state. The archiving mode is usually set during database creation, but it can be reset at a later stage.

We can recover a database damaged by a media failure in one of three ways after you have restored backups of the damaged data files. These steps can be

performed using the Server Manager's Apply Recovery Archives dialog box, using the Server Manager's RECOVER command, or using the SQL ALTER DATABASE command :

- We can recover an entire database using the RECOVER DATABASE command. This command performs media recovery on all of the data files that require redo processing.
- We can recover specified tablespaces using the RECOVER TABLESPACE command. This command performs media recovery on all of the data files in the listed tablespaces. Oracle requires the database to be open and mounted in order to determine the file names of the tables contained in the tablespace.
- We can list the individual files to be recovered using the RECOVER DATAFILE command. The database can be open or closed, provided that Oracle can take the required media recovery locks.

In certain situations, we can also recover a specific damaged data file, even if a backup file isn't available. This can only be done if all of the required log files are available and the control file contains the name of the damaged file. In addition, Oracle provides a variety of recovery options for different crash scenarios, including incomplete recovery, change-based, cancel-based, and time-based recovery, and recovery from user errors.

Prevention is Better than Cure :

Although each DBMS we reviewed has a range of backup and recovery facilities, it is always important to ensure that the facilities are used properly and adequately. By "adequately," I mean that backups must be taken regularly. All of the DBMSs we reviewed provided the facilities to repost or re-execute completed transactions from a log or journal file. However, reposting or re-executing a few weeks' worth of transactions may take an unbearably long time. In many situations, users require quick access to their databases, even in the presence of media failures. Remember that the end users are not concerned with physical technicalities, such as restoring a database after a system crash

Even better than quick recovery is no recovery, which can be achieved in two ways. First, by performing adequate system monitoring and using proper procedures and good equipment, most system crashes can be avoided. It is better to provide users with a system that is up and available 90 percent of the time than to have to do sporadic fixes when problems occur. Second, by

using redundant databases such as hot standby or replicated databases, users can be relieved of the recovery delays: Users can be switched to the hot backup database while the master database is being recovered.

A last but extremely important aspect of backup and recovery is testing. Test your backup and recovery procedures in a test environment before deploying them in the production environment. In addition, the backup and recovery procedures and facilities used in the production environment must also be tested regularly. A recovery scheme that worked perfectly well in a test environment is useless if it cannot be repeated in the production environment particularly in that crucial moment when the root disk fails during the month-end run!

□ □

Chapter-12

Database Design

Q.1 Define Functional Dependencies.

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

Trivial Functional Dependency : A trivial functional dependency is a functional dependency of an attribute on a superset of itself. {Employee ID, Employee Address} → {Employee Address} is trivial, as is {Employee Address} → {Employee Address}.

Full Functional Dependency : An attribute is fully functionally dependent on a set of attributes X if it is functionally dependent on X, and not functionally dependent on any proper subset of X. {Employee Address} has a functional dependency on {Employee ID, Skill}, but not a *full* functional dependency, because is also dependent on {Employee ID}.

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

Multivalued Dependency : A multivalued dependency is a constraint according to which the presence of certain rows in a table implies the presence of certain other rows: see the Multivalued Dependency article for a rigorous definition.

Join Dependency : A table *T* is subject to a join dependency if *T* can always be recreated by joining multiple tables each having a subset of the attributes of *T*.

Non-PrimeAttribute : A non-prime attribute is an attribute that does not occur in any candidate key. Employee Address would be a non-prime attribute in the "Employees' Skills" table.

Q.2 What is Normalization? Explain it with the problems addressed by Normalization.

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

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

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

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

So, we can say, Normalization is an essential process of database design. A good understanding of the semantics of data helps the designer to build efficient design in using the concept of normalization. In other words, "Normalization is a process in which we can reduce redundancy and in consistency in our relation."

Q. 3. Write in brief purpose of normalization.

Ans. Purpose of normalization :

Minimize redundancies in data.

Remove insert, delete and update anomalies during database activity.

Reduce the need to reorganize data when it is modified or enhanced.

Q. 4. What are the Problems oranomalies Addressed by Normalization?

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

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

answers to the question of what this particular employee's address is. This phenomenon is known as an **update anomaly**.

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

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

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

Q.5 Explain 1NF, 2NF, 3NF, BCNF, 4NF and PJNF.

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

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

History :

Edgar F. Codd first proposed the process of normalization and what came to be known as the **1st Normal Form**.

There is, in fact, a very simple elimination procedure which we shall call normalization. Through decomposition non-simple domains are replaced by "*domains whose elements are atomic (non-decomposable) values.*"

—Edgar F. Codd, A Relational Model of Data for Large Shared Data Banks.

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

Normal Forms :

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

The normal forms are applicable to individual tables; to say that an entire database is in normal form *n* is to say that all of its tables are in normal form *n*.

Newcomers to database design sometimes suppose that normalization proceeds in an iterative fashion, i.e. a 1NF design is first normalized to 2NF, then to 3NF, and so on. This is not an accurate description of how normalization typically works. A sensibly designed table is likely to be in 3NF on the first attempt; furthermore, if it is 3NF, it is overwhelmingly likely to have an HNF of 5NF. Achieving the "higher" normal forms (above 3NF) does not usually require an extra expenditure of effort on the part of the designer, because 3NF tables usually need no modification to meet the requirements of these higher normal forms.

Edgar F. Codd originally defined the first three normal forms (1NF, 2NF, and 3NF). These normal forms have been summarized as requiring that all non-key attributes be dependent on "the key, the whole key and nothing but the key". The fourth and fifth normal forms (4NF and 5NF) deal specifically with

the representation of many-to-many and one-to-many relationships among attributes. Sixth normal form (6NF) incorporates considerations relevant to temporal databases.

First Normal Form :

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

Note that the restriction on nullable columns as a 1NF requirement, as espoused by Chris Date, et. al., is controversial. This particular requirement for 1NF is a direct contradiction to Dr. Codd's vision of the relational database, in which he stated that "null values" must be supported in a fully relational DBMS in order to represent "missing information and inapplicable information in a systematic way, independent of data type." By redefining 1NF to exclude nullable columns in 1NF, no level of normalization can ever be achieved unless all nullable columns are completely eliminated from the entire database. This is in line with Date's and Darwen's vision of the perfect relational database, but can introduce additional complexities in SQL databases to the point of impracticality.

In 1NF, requirement of a relation is that every table contain exactly one value for each attribute. This is sometimes expressed as "no repeating groups". While that statement itself is axiomatic, experts disagree about what qualifies as a "repeating group", in particular whether a value may be a relation value; thus the precise definition of 1NF is the subject of some controversy. Notwithstanding, this theoretical uncertainty applies to relations, not tables. Table manifestations are intrinsically free of variable repeating groups because they are structurally constrained to the same number of columns in all rows. Therefore "a relation is in 1NF if its each attribute is having atomic value."

Second Normal Form :

The criteria for **Second Normal Form** (**2NF**) are :

—The table must be in 1NF.

None of the non-prime attributes of the table are functionally dependent on a part (proper subset) of a candidate key;‖ in other words, all functional dependencies of non-prime attributes on candidate keys are full functional dependencies. For example, consider an "Employees' Skills" table whose attributes are Employee ID, Employee Address, and Skill; and suppose that the combination of Employee ID and Skill uniquely identifies records within the table. Given that Employee Address depends on only one of those attributes – namely, Employee ID – the table is not in 2NF.

In simple, **"a table is in 2NF if it is in 1NF and all non-prime attributes/columns are fully dependent on Primary Key(s), or a relation is in 2NF if it is in 1NF and every non-key attribute is fully dependent on each candidate key of the relation."**

Note that if none of a 1NF table's candidate keys are composite – i.e. every candidate key consists of just **one** attribute – then we can say immediately that the table is in 2NF.

Third Normal Form :

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

"The table must be in 2NF.

Every non-prime attribute of the table must be non-transitively dependent on each candidate key." A violation of 3NF would mean that at least one non-prime attribute is only *indirectly* dependent (transitively dependent) on a candidate key. For example, consider a "Departments" table whose attributes are Department ID, Department Name, Manager ID, and Manager Hire Date; and suppose that each manager can manage one or more departments. {Department ID} is a candidate key. Although Manager Hire Date is functionally dependent on the candidate key {Department ID}, this is only because Manager Hire Date depends on Manager ID, which in turn depends on Department ID. This transitive dependency means the table is not in 3NF.

Boyce-Codd Normal Form :

A table is in **Boyce-Codd Normal Form** (**BCNF**) if and only if, for every one of its non-trivial functional dependencies $X \rightarrow Y$, X is a superkey—that is, X is either a candidate key or a superset thereof.

EMP Table

Á Â Ã

EMP NO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	Å	17-NOV-81	5000		10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUNE-81	2450		10
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	Ä 0	30
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7934À	MILLER	CLERK	7782	23-JAN-82	1300		10

Fourth Normal Form :

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

Fifth Normal Form :

The criteria for **Fifth Normal Form** (**5NF** and also **PJ/NF**) are -

—The table must be in 4NF.

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

Domain/Key Normal Form :

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

Sixth Normal Form :

A table is in Sixth Normal Form (6NF) if and only if it satisfies no non-trivial join dependencies at all, meaning that the fifth normal form is also satisfied. The sixth normal form was only defined when extending the relational model to take into account the temporal dimension. Most SQL technologies, as of 2005, do not take into account this work, and most temporal extensions to SQL are not relational. See work by Date, Darwen and Lorentzos worked for a relational temporal extension, Zimyani for further discussion on Temporal Aggregation in SQL, or TSQL2 for a non-relational approach.

Domain Key Normal Form (DKNF)

A relation schema R is said to be in DKNF if all constraints and dependencies required are enforced simply by enforcing the domain constraints and key constraints on the relation.

The idea behind domain-key normal form is to specify the 'Ultimate normal form' that takes into account all possible types of dependencies and constraints. Before defining DKNF formally let us define few types of constraints.

Domain Constraint: It specifies that each attribute, Ai of relation R (A_1, A_2 .... A_n), must have a value from a set SAi. Domain Constraint on attribute Ai is specified as IN (Ai, SAi) we have used domain constraints as a part of integrity constraints.

Key Constraint: For the relation schema R (A_1, A_2 A_n), the Key Constraint, KEY (K], where K is a subset of R, is the restriction that no two tuples of relation r defined on 'the relation schema R have the same value for the attributes in K.

General Constraint: A general constraint is expressed as a simple statement or predicate and specifies some special requirement that each tuple of a relation must satisfy to be a valid tuple. For an example or general constraint consider the relation catalog (pid, sid, cost) shown in figure 6.23 where pid is the id of product~ sid is the id of supplier and cost is the cost of product supplied by a supplier suppose we have constraint that if cost of a product is more than Rs. 5000 supplier will be only A or C otherwise it may be one of A, B, C, D or E. This constraint can not be expressed as domain constraint or a key constraint and hence has to be categorized as general constraint.

A relation schema R is said to be in DKNF if all constraints and dependencies that should hold on a valid relation state can be enforced simply by enforcing the domain constraints and key constraints on the relation. However because it is difficult to specify complex constraints in form of domain or key constraints so its practical utility is limited.

P	Si	Co
1	D	200
1	B	300
1	C	550
1	E	400
10	A	5800

Figure 6.23 : Instance of Catalog relation

The relation catalog can be convel1ed in DKNF by decomposing it two relations Low cost catalog and High cost catalog. Low cost catalog relation will contain products for which cost is less than Rs. 5000 and domain of Sid will be {A, B, C, D, E}. The other relation high cost catalog will contain products for which cost is more than Rs. 5000 and domain of Sid will be {A, C}. Now the general constraint is converted into domain constraints.

Q.6 Define Denomalization in Databases.

Ans.: Denormalization : In some exceptional cases, database designers use the redundancy to improve performance for specific applications.They select a schema which is having duplicate values that means it is not normalized. for ex., suppose that the name of an account holder, has to be displayed along with the acc. no. and balance, every time the account accessed. In our normalized schema, this requires join of account with depositor. One alternative is to join account with depositor, which creates all the attributes of both the relation.This makes displaying the info. faster. However the balance is repeated for every person who owns the account, and all copies must be updated by the application,whenever the account balance is updated. This process of taking normalized schema and making it non-normalize is called denormalization.

Databases intended for Online Transaction Processing (OLTP) are typically more normalized than databases intended for Online Analytical Processing

(OLAP). OLTP Applications are characterized by a high volume of small transactions such as updating a sales record at a super market checkout counter. The expectation is that each transaction will leave the database in a consistent state. By contrast, databases intended for OLAP operations are primarily "read mostly" databases. OLAP applications tend to extract historical data that has accumulated over a long period of time. For such databases, redundant or "denormalized" data may facilitate Business Intelligence applications. Specifically, dimensional tables in a star schema often contain denormalized data. The denormalized or redundant data must be carefully controlled during ETL processing, and users should not be permitted to see the data until it is in a consistent state. The normalized alternative to the star schema is the snowflake schema. It has never been proven that this denormalization itself provides any increase in performance, or if the concurrent removal of data constraints is what increases the performance. In many cases, the need for denormalization has waned as computers and RDBMS software have become more powerful, but since data volumes have generally increased along with hardware and software performance, OLAP databases often still use denormalized schemas.

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

Non-First Normal Form (NF^2 or N1NF) :

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

Consider the following table :

Non-First Normal Form	
Person	**Favorite Colors**
Bob	blue, red
Jane	green, yellow, red

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

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

□ □ □

Chapter-13

Database Tuning

Q.1 What are views in DBMS?

Ans.: Views :

We have assumed up to now that the relations we are given are the actual relations stored in the database.

For security and convenience reasons, we may wish to create a personalized collection of relations for a user.

We use the term **view** to refer to any relation, not part of the conceptual model, that is made visible to the user as a ``virtual relation''.

As relations may be modified by deletions, insertions and updates, it is generally not possible to store views. (Why?) Views must then be recomputed for each query referring to them.

View Definition :

A view is defined using the **create view** command :

create view v as < query expression >

where <query expression> is any legal query expression.

The view created is given the name v

To create a view *all-customer* of all branches and their

customers: **create view all-customer as**

$\prod_{bname, ename}$ **(deposit) U** $\prod_{bname, ename}$ **(borrow)**

Having defined a view, we can now use it to refer to the virtual relation it creates. View names can appear anywhere a relation name can.

We can now find all customers of the SFU branch by writing

$ename\ (\sigma_{bname = \text{“SFU”}}(all - customer))$

Updates Through Views and Null Values :

Updates, insertions and deletions using views can cause problems. The modifications on a view must be transformed to modifications of the actual relations in the conceptual model of the database.

An example will illustrate: consider a clerk who needs to see all information in the *borrow* relation except *amount*.

Let the view *loan-info* be given to the clerk:

create view loan-info as

$\prod_{\text{bname , loan\#, ename}}$ **(borrow)**

Since SQL allows a view name to appear anywhere a relation name may appear, the clerk can write:

loan-info ← loan-info U {("SFU",3,"Ruth")}

This insertion is represented by an insertion into the actual relation *borrow*, from which the view is constructed.

However, we have no value for *amount*. A suitable response would be

Reject the insertion and inform the user.

Insert (``*SFU*'',3,``*Ruth*'',null) into the relation.

The symbol **null** represents a null or place-holder value. It says the value is unknown or does not exist.

Another problem with modification through views: consider the

view **create view branch-city as**

$\prod_{\text{bname , ecity}}$ **(borrow x customer)**

This view lists the cities in which the borrowers of each branch live.

Now consider the insertion

branch-city ← branch-city U {("Brighton","Woodside")}

Using nulls is the only possible way to do this (see Figure 3.22 in the textbook).

If we do this insertion with nulls, now consider the expression the view actually corresponds to:

$\prod_{\text{bname , ecity}}$ **(borrow x customer)**

As comparisons involving nulls are always **false**, this query misses the inserted tuple.

To understand why, think about the tuples that got inserted into *borrow* and *customer*. So we can think about how the view is recomputed for the above query.

Q.2 What do you mean by Tuning? Also define Tuning Indexes in brief.

Ans.: Database Tuning describes a group of activities used to optimize and homogenize the performance of a database. It usually overlaps with query tuning, but refers to configuration of the database files, the database management system (DBMS), and the operating system and hardware the DBMS runs on.

The goal is to maximize use of system resources to perform work as efficiently and rapidly as possible. Most systems are designed to manage work efficiently, but it is possible to greatly improve performance by customizing settings and the configuration for the database and the DBMS being tuned.

Index tuning as part of database tuning is the task of selecting and creating indexes with the goal of reducing query processing times. However, in dynamic environments with various ad-hoc queries it is difficult to identify potentially useful indexes in advance. So that need for new indexing schemes suitable for self-tuning. Based on problems with previous approaches we describe the key concepts, which are sparse and partial indexing, usage-balanced instead of data-balanced structures, and dynamic resource assignment. We illustrate the approach by a simple index structure, which provides adaptability as well as improved access characteristics for indexing in this manner.

Q.3 Explain DBMS Benchmarking.

Ans.: In computing, a **Benchmark** is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it. The term, benchmark, is also commonly used for specially-designed benchmarking programs themselves. Benchmarking is usually associated with assessing performance characteristics of computer hardware, for example, the floating point operation performance of a CPU, but there are circumstances when the technique is also applicable to software. Software

benchmarks are, for example, run against compilers or database management systems.

□ □ □

Chapter-14

Database Security & its implementation

Q.1 Explain Security in DBMS and Its Implimentation.

Ans. : Data Security actually is an important function of a database management system whether it is centralized or distributed. Data security controls protects the data from unauthorized access and unwanted changes.

Data security controls have two major aspects :

Data Protection

Authorization Control

Data protections make sure that no any unauthorized person can understand the physical contents of data.

File systems in centralized and distributed operating systems are used to provide this type of security. Mostly used approach for providing this protection is data encryption.

Data encryption encodes the data such that nobody can understand the actual data contents. His encryption not only useful to secure the data stored on disks but also for exchanging the information over a network.

This encoded data can be decoded (decrypted) only by than authorized users that know what the code is. Authorization security control ensures that only privileged user can manipulate the data in the way they are allowed to do. The database management system must determine that which users are allowed to perform which functions and which data portion is accessible by them.

Authorization controls are different in a centralized database to the distributed database environment. Authorization control definition in a distributed database system is derived from that in centralized system but in

the context of distributed system some additional complexity is also considered.

Q.2 Define various Access Controls in terms of Database Security.

Ans.: Access Control is the ability to permit or deny the use of a particular resource by a particular entity. Access control mechanisms can be used in managing physical resources (such as a movie theater, to which only ticketholders should be admitted), logical resources (a bank account, with a limited number of people authorized to make a withdrawal), or digital resources (for example, a private text document on a computer, which only certain users should be able to read).

Access control techniques are sometimes categorized as either discretionary or non-discretionary. The three most widely recognized models are Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC). MAC and RBAC are both non-discretionary.

Discretionary Access Control :

Discretionary Access Control (DAC) is an access policy determined by the owner of an object. The owner decides who is allowed to access the object and what privileges they have.

Two important **concepts in DAC** are :

- **File and Data Ownership :** Every object in the system has an owner. In most DAC systems, each object's initial owner is the subject that caused it to be created. The access policy for an object is determined by its owner.
- **Access Rights and Permissions :** These are the controls that an owner can assign to other subjects for specific resources.

Access controls may be discretionary in ACL-based or capability-based access control systems. (In capability-based systems, there is usually no explicit concept of 'owner', but the creator of an object has a similar degree of control over its access policy.)

Mandatory Access Control :

Mandatory Access Control (MAC) is an access policy determined by the system, not the owner. MAC is used in multilevel systems that process highly sensitive data, such as classified government and military information. A multilevel system is a single computer system that handles multiple classification levels between subjects and objects.

- **Sensitivity labels :** In a MAC-based system, all subjects and objects must have labels assigned to them. A subject's sensitivity label specifies its level of trust. An object's sensitivity label specifies the level of trust required for access. In order to access a given object, the subject must have a sensitivity level equal to or higher than the requested object.
- **Data Import and Export :** Controlling the import of information from other systems and export to other systems (including printers) is a critical function of MAC-based systems, which must ensure that sensitivity labels are properly maintained and implemented so that sensitive information is appropriately protected at all times.

Two methods are commonly used for applying mandatory access control :

- **Rule-Based Access Controls :** This type of control further defines specific conditions for access to a requested object. All MAC-based systems implement a simple form of rule-based access control to determine whether access should be granted or denied by matching:

  An object's Sensitivity Label

  A subject's Sensitivity Label

Q.3 What do you mean by Encryption?

Ans.: In Cryptography, **Encryption** is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is encrypted information (in cryptography, referred to as ciphertext). In many contexts, the word encryption also implicitly refers to the reverse process, decryption (e.g. ―software for encryption‖ can typically also perform decryption), to make the encrypted information readable again (i.e. to make it unencrypted).

Encryption has long been used by militaries and governments to facilitate secret communication. Encryption is now used in protecting information within many kinds of civilian systems, such as computers, networks (e.g. the

Internet e-commerce), mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. Encryption is also used in digital rights management to prevent unauthorized use or reproduction of copyrighted material and in software also to protect against reverse engineering (see also copy protection).

Encryption, by itself, can protect the confidentiality of messages, but other techniques are still needed to protect the integrity and authenticity of a message; for example, verification of a message authentication code (MAC) or a digital signature. Standards and cryptographic software and hardware to perform encryption are widely available, but successfully using encryption to ensure security may be a challenging problem. A single slip-up in system design or execution can allow successful attacks. Sometimes an adversary can obtain unencrypted information without directly undoing the encryption. See, e.g., traffic analysis, TEMPEST, or Trojan horse.

This question paper contains 2 printed pages. Roll No. .....

B.C.A. (Pt. - II) **Data. Mana. Sys.**

204/234

B.C.A. (Part - II) EXAMINATION, 2021

(Faculty of Science)

(Three - Year Scheme of 10+2+3 Pattern)

DATABASE MANAGEMENT SYSTEM

Time Allowed : Three Hours Maximum Marks : 100

No supplementary answer-book will be given to any candidate. Hence the candidates should write their answers precisely in the main answer-book only.

All the parts of one question should be answered at one place in the answer-book. One complete question should not be answered at different places in the answer-book.

Write your roll number on question paper before start writing answers of questions.

*Question paper consists of **three** parts. **All three** parts are **compulsory**.*

PART - A : *(Very short answer) consists of **10** questions of **two** marks each. Maximum limit for each question is upto **40** words.*

PART - B : *(Short answer) consists of **5** questions of **four** marks each. Maximum limit for each question is upto **80** words.*

PART - C : *(Long answer) consists of **5** questions of **twelve** marks each with an internal choice.*

PART - A

1. (a) Define database. **10x2=20**
   (b) What is mean by data independance ?
   (c) Define attributes and entities.
   (d) What is mean by aggregation ?
   (e) What is mean by data recovery ?
   (f) Define transactions.
   (g) What is SQL ?
   (h) Define views.
   (i) Define distributed database.
   (j) What is mean by object database ?

PART - B

2. (a) Discuss the Database system v/s File system. **5x4=20**
   (b) Discuss the various types of keys.
   (c) Explain Boyce Codd Normal Form with examples.
   (d) Discuss the Aggregate functions with examples.
   (e) What is mean by concurrency control ? Discuss the concurrency control in distributed databases.

PART - C

3. (a) Discuss the architecture of DBMS. **6+6**
   (b) Discuss the role of database administrator.

OR

   (a) Explain the advantages and disadvantages of DBMS. **10+2**
   (b) What is mean by data independance ?

4. (a) Discuss the fundamental operations of relational algebra. 8+4
(b) Discuss the generalization and aggregation.

OR

Write short notes on : 4+8
(a) Mapping constraints
(b) E-R Model

5. Discuss the functional dependencies, access control, backup, recovery and maintenance. 12

OR

Discuss the various Normal Forms with examples. 12

6. (a) Explain SQL Data types. 5+7
(b) Discuss the insert, update and delete operations with examples.

OR

Write short notes on : 4+4+4
(a) Characteristics of SQL
(b) Types of SQL commands
(c) Join, Union and Intersection in SQL

7. (a) Explain object oriented databases. 6+6
(b) Discuss Distributed Query Processing.

OR

Write short notes on : 6+3+3
(a) Object Oriented data model
(b) Object Relational databases
(c) Distributed Transactions

- o O o -

This question paper contains 3 printed pages] Roll No

Sl.No. **234**

B.C.A. (Part. II)

B.C.A. (Part - II) EXAMINATION, 2017
(Faculty of Science)
(Three-Year Scheme of 10 + 2+ 3 Pattern)
Paper - 234
DATABASE MANAGEMENT SYSTEM

Time : Three Hours] *[Maximum Marks : 100*

Answer of all the questions (short answer as well as descriptive) are to be given in the main answer -book only. Answers of short answer type questions must be given in sequential order. Similarly all the parts of one question of descriptive part should be answered at one place in the answer-book. One complete question should not be answered at different places in the answer-book. Write your roll numbers on question paper before start writing answers of questions.

Part I: *(Very short Answer) consists of 10 questions of two marks each. Maximum limit for each question is up to 40 words.*

Part II: *(Short answer) consists of 5 questions of four marks each. Maximum limit for each question is up to 80 words.*

Part III: *(Long answer) consists of 5 questions of twelve marks each with internal choice.*

PART - I

Q1) a) List any four advantages of using a DBMS.

b) What is a schema?

c) What is an E-R Diagram?

d) Differentiate between super key and candidate key.

e) What is a transaction?

f) What is Join statement?

P.T.O.

g) What is sub query?

h) Why normalization is required?

i) How transaction is committed in a distributed system?

j) What is ODL?

PART - II

Q2) a) Differentiate between Logical and Physical data independence. **[4]**

b) What is aggregation? Give example. **[4]**

c) Explain the structure of SQL SELECT statement. **[4]**

d) What is access control? **[4]**

e) What do you mean by concurrency control? **[4]**

PART - III

Q3) Define DBMS. What are the advantages of DBMS over conventional file processing system? What are the functions of a database administrator? **[2+6+4]**

OR

Explain the rules defined by Codd that are necessary for any DBMS to be considered as a RDBMS. **[12]**

Q4) Differentiate between following: **[3×4=12]**

a) Strong and Weak entity

b) Referential and Domain integrity

c) Single valued and multi valued attributes

OR

What is relational algebra? Explain different types of join and aggregate operations of relational algebra. Give appropriate examples.

[2+10]

Q5) Explain normalization & its different forms. Give appropriate examples. **[12]**

OR

Describe backup and recovery mechanisms available in DBMS.

Q6) Write SQL queries for the following: **[12]**

Consider the table SPORTS having fields: (RollNo, Class, Name, Game, Grade)

a) Display the names of the students who have grade 'C' or grade 'D'.

b) Display the grade of the students whose name starts with 'D'

c) Display the different games offered.

d) Display the Roll number and name of the student who belong to class '7' and plays hockey.

e) Delete the student record whose roll no. is 101.

OR

Explain the data types availabale in SQL. Also explain various aggregate functions in SQL with suitable examples.

Q7) Explain the following: **[12]**

a) Distributed Transactions

b) Object-relational Databases

OR

Define object databases. Describe persistent programming languages. What are the several approaches proposed to make the objects persistent?

This question paper contains 2 printed pages.

B.C.A. (Part - II)

234 **Data. Mana. Sys.**

B.C.A. (PART II) EXAMINATION - 2018

(Faculty of Science)

(Three-year Scheme of 10 + 2 + 3 Pattern)

Paper - 234

(Database Management System)

Time allowed : Three Hours

Maximum Marks : 100

PART - I: (Very Short Answer) consists of 10 questions of 2 marks each. Maximum limit for each question is up to 40 words.

PART - II: (Short answer) consists of 5 questions of 4 marks each. Maximum limit for each question is up to 80 words.

PART - III: (Long answer) consists of 5 questions of 12 marks each with internal choice.

PART - I

1. (a) Give any four drawback of file System.
   (b) What is mean by data independance?
   (c) What is mean by candidate Key?
   (d) Define Schema.
   (e) What is mean by Transactions?
   (f) Define Access control.
   (g) Define views.
   (h) What is mean by Aggregate functions?
   (i) Define concurrency control.
   (j) Give four features of Object Oriented Databases. [10 x 2 = 20]

PART - II

2. (a) Discuss the architecture of DBMS.
   (b) Write a note on Generalization and aggregation.
   (c) What is mean by boyce codd NF? Explain.
   (d) Write a note on types of SQL commands and SQL operators.
   (e) Write a note on Object-Relational Databases. [5 x 4 = 20]

PART - III

3 (a) Discuss the advantages and disadvantages of DBMS.

(b) Define Instances. 10+2

OR

Write short notes on-

(i) Database Administrator

(ii) Data Base System v/s File System 6+6

4. (a) Discuss Entity Relational Model with example.

(b) What is mean by mapping constraints? 9+3

OR

Write short notes on-

(i) Operations of Relational Algebra

(ii) Keys 6+6

5. (a) Explain 1st, 2nd and 3rd normal forms with example.

(b) What is mean by Backup and Recovery? 8+4

OR

Write short notes on-

(i) Functional Dependencies

(ii) Transactions & their states.

(iii) Loss Less decomposition 4+4+4

6. (a) Discuss the characteristics and advantages of SQL.

(b) What is mean by minus in SQL? 10+2

OR

Write short notes on-

(i) SQL Data types

(ii) Aggregate Functions

(iii) Tables and Indexes 4+4+4

7. (a) Discuss Distributed Query Processing.

(b) Explain Object-Oriented Data Model. 6+6

OR

Write short notes on-

(i) Object-Oriented Databases

(ii) Distributed Transactions

(iii) Persistend Programming Languages 4+4+4