SHRIMATI INDIRA GANDHI COLLEGE

Affiliated to Bharathidasan University
Nationally Accredited at 'A' Grade(3rd Cycle) by NAAC
An ISO 9001:2015 Certified Institution

Thiruchirrappalli

ADVANCED DATABASE MANAGEMENT SYSTEMS

(P22ITCC12)

STUDY MATERIAL

DEPARTMENT OF COMPUTER SCIENCE, INFORMATION TECHNOLOGY AND COMPUTER APPLICATIONS

Prepared by

Ms.P. ANANTHI, Ms.V.VETRISELVI

Assistant Professors in Computer Science

SHRIMATI INDIRA GANDHI COLLEGE

TIRUCHIRAPPALLI – 2.

ADVANCED DATABASE MANAGEMENT SYSTEMS

(P22ITCC12)

UNIT-1

Database: File Processing System Vs DBMS, History, Characteristic-Abstraction levels, Architecture of a database, Functional components of a DBMS, DBMS Languages-Database users and DBA.

File Processing System Vs DBMS

A database management system coordinates both the physical and the logical access to the data, whereas a file-processing system coordinates only the physical access.

A database management system is designed to allow flexible access to data (i.e. queries), whereas a file-processing system is designed to allow predetermined access to data (i.e. compiled programs).

A database management system is designed to coordinate multiple users accessing the same data at the same time. A file-processing system is usually designed to allow one or more programs to access different data files at the same time. In a file-processing system, a file can be accessed by two programs concurrently only if both programs have read-only access to the file.

Redundancy is control in DBMS, but not in file system.

Unauthorized access is restricted in DBMS but not in the file system.

DBMS provide backup and recovery whereas data lost in file system can't be recovered.

DBMS provide multiple user interfaces. Data is isolated in file system.

DBMS	File Processing System
Minimal data redundancy problem in DBMS	Data Redundancy problem exits
Data Inconsistency does not exist	Data Inconsistency exist here
Accessing database is easier	Accessing is comparatively difficult
The problem of data isolation is not found in database	Data is scattered in various files and files may be of different format, so data isolation problem exists
Transactions like insert, delete, view, updating, etc are possible in database	In file system, transactions are not possible
Concurrent access and recovery is possible in database	Concurrent access and recovery is not possible
Security of data	Security of data is not good
A database manager (administrator) stores the relationship in form of structural tables	A file manager is used to store all relationships in directories in file systems.

History of Database

1950s and early 1960s:

- Data processing using magnetic tapes for storage
- Tapes provided only sequential access
- Punched cards for input

Late 1960s and 1970s:

- Hard disks allowed direct access to data
- Hierarchical and network data models in widespread use
  - IBM's DL/I (Data Language One)
  - CODAYSL's DBTG (Data Base Task Group) model → the basis of current DBMSs
- Ted Codd defines the relational data model
  - IBM Research develops System R prototype
  - UC Berkeley develops Ingres prototype

Entity-Relationship Model for database design 1980s:

- Research relational prototypes evolve into commercial systems
  - DB2 from IBM is the first DBMS product based on the relational model
  - Oracle and Microsoft SQL Server are the most prominent commercial DBMS products based on the relational model
- SQL becomes industrial standard
- Parallel and distributed database systems
- Object-oriented database systems (OODBMS)
  - Goal: store object-oriented programming objects in a database without having to transform them into relational format
  - In the end, OODBMS were not commercially successful due to high cost of relational to object-oriented transformation and a sound underlying theory, but they still exist
- Object-relational database systems allow both relational and object views of data in the same database

Late 1990s:

- Large decision support and data-mining applications
- Large multi-terabyte data warehouses
- Emergence of Web commerce

Early 2000s:

- XML and XQuery standards
- Automated database administration

Later 2000s:

- Web databases (semi-structured data, XML, complex data types)
- Cloud computing
- Giant data storage systems (Google BigTable, Yahoo PNuts, AmazonWeb Services, …)

Characteristics of a Database

Stores any kind of Data

A database management system should be able to store any kind of data. It should not be restricted to the employee name, salary and address. Any kind of data that exists in the real world can be stored in DBMS because we need to work with all kinds of data that is present around us.

Support ACID Properties

Any DBMS is able to support ACID (Accuracy, Completeness, Isolation, and Durability) properties. It is made sure is every DBMS that the real purpose of data should not be lost while performing transactions like delete, insert an update. Let us take an example; if an employee name is updated then it should make sure that there is no duplicate data and no mismatch of student information.

Represents complex relationship between data

Data stored in a database is connected with each other and a relationship is made in between data. DBMS should be able to represent the complex relationship between data to make the efficient and accurate use of data.

Backup and recovery

There are many chances of failure of whole database. At that time no one will be able to get the database back and for sure company will be in a big loss. The only solution is to take backup of database and whenever it is needed, it can be stored back. All the databases must have this characteristic.

Structures and described data

A database should not contains only the data but also all the structures and definitions of the data. This data represent itself that what actions should be taken on it. These descriptions include the structure, types and format of data and relationship between them.

Data integrity

This is one of the most important characteristics of database management system. Integrity ensures the quality and reliability of database system. It protects the unauthorized access of database and makes it more secure. It brings only the consistence and accurate data into the database.

Concurrent use of database

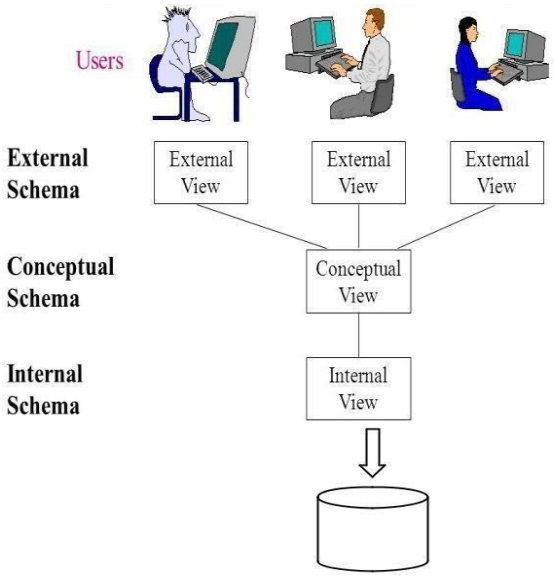
There are many chances that many users will be accessing the data at the same time. They may require altering the database system concurrently. At that time, DBMS supports them to concurrently use the database without any problem.

Abstraction levels

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:



Levels of Abstraction in a DBMS

- *Physical level (or Internal View / Schema):* The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.
- *Logical level (or Conceptual View / Schema):* 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 the 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.

- This is referred to as **physical data independence**.
- ***View level (or External View / Schema):*** 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 the 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.

For example, we may describe a record as follows:

```
type instructor = record
        ID : char (5);
        name : char (20);
        dept name : char (20);
        salary : numeric (8,2);
end;
```

This code defines a new record type called *instructor* with four fields. Each field has a name and a type associated with it. A university organization may have several such record types, including

department, with fields *dept_name*, *building*, and *budget*

course, with fields *course_id*, *title*, *dept_name*, and *credits*

student with fields ID, name, dept_name and tot_cred

- At the physical level, an *instructor*, *department*, or *student* record can be described as a block of consecutive storage locations.
- At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined

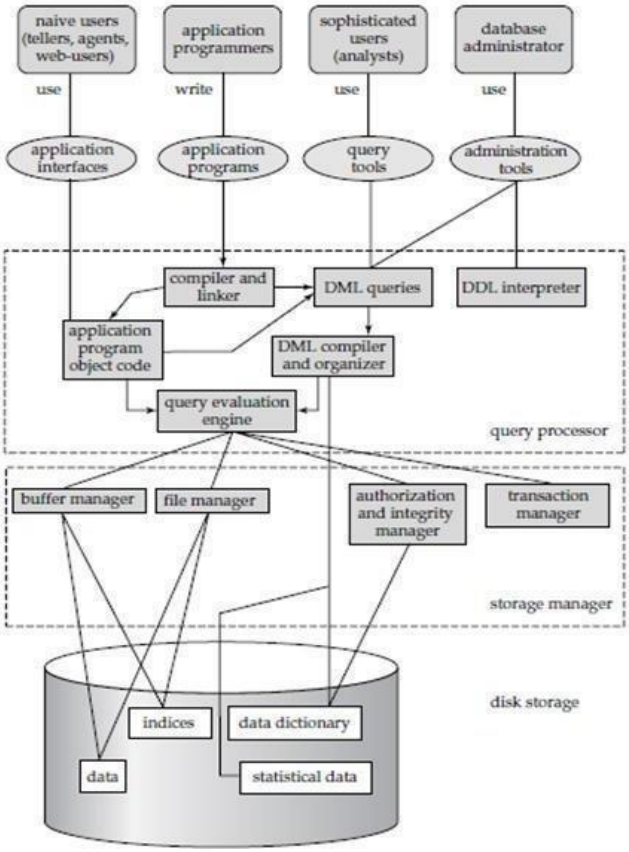
as well.

- Finally, at the view level, computer users see a set of application programs that hide details of the data types. At the view level, several views of the database are defined, and a databaseuser sees some or all of these views.

Architecture of a Database

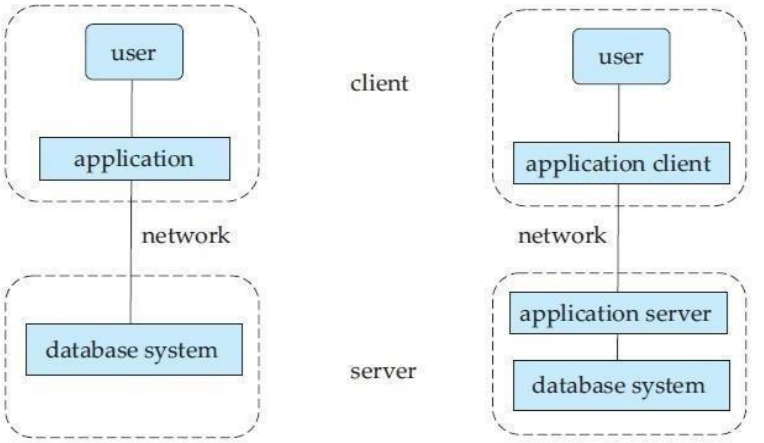
The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client- server, where one server machine executes work on behalf of multiple client machines.

Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines.



A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly

divided into the storage manager and the query processor components. The storage manager is important because databases typically require a large amount of storage space. The query processor is important because it helps the database system simplify and facilitate access to data.



(a) Two-tier architecture (b) Three-tier architecture

Query Processor:

The query processor components include

- **DDL interpreter,** which interprets DDL statements and records the definitions in the data dictionary.

DML compiler, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands. A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs **query optimization**, that is, it picks the lowest cost evaluation plan from among the alternatives.

Query evaluation engine, which executes low-level instructions generated by the DMLcompiler.

Storage Manager:

A storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager.

Transaction Manager:

A **transaction** is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database-consistency constraints.

Components of a Database

User: - Users are the one who really uses the database. Users can be administrator, developer or the end users.

Data or Database: - As we discussed already, data is one of the important factor of database. A very huge amount of data will be stored in the database and it forms the main source for all other components to interact with each other. There are two types of data. One is user data. It contains the data which is responsible for the database, i.e.; based on the requirement, the data will be stored in the various tables of the database in the form of rows and columns. Another data is Metadata. It is known as 'data about data', i.e.; it stores the information like how many tables, their names, how many columns and their names, primary keys, foreign keys etc. basically these metadata will have information about each tables and their constraints in the database.

DBMS: - This is the software helps the user to interact with the database. It allows the users to insert, delete, update or retrieve the data. All these operations are handled by query languages like MySQL, Oracle etc.

Database Application: - It the application program which helps the users to interact with the database by means of query languages.

Database application will not have anyidea about the underlying DBMS.

DBMS Languages

To read data, update and store information in DBMS, some languages are used. Databaselanguages in DBMS are given as below.

- DDL – Data Definition Language
- DML – Data Manipulation Language
- DCL – Data Control Language
- TCL – Transaction Control Language

1. Data Definition Language (DDL)

DDL stands for data definition language and used to define database patterns or structures. DDL is a syntax which is same as syntax of computer programming language for definingpatterns of database.

Few examples of it are:

- CREATE – used to create objects in database
- ALTER – alter the pattern of database
- DROP – helps in detecting objects
- TRUNCATE – erase all records from table
- COMMENT – adding of comments to data dictionary
- RENAME – useful in renaming an object

CREATE statement or command is used to create a new database. In structured query language the create command creates an object in a relational database management system. The commonly used create command is as follows

- CREATE TABLE [name of table] ([definitions of column]) [parameters of table]

DROP statement destroys or deletes database or table. In structured query language, it also deletes an object from relational database management system. Typically used DROP statement is

- DROP type of object name of object

ALTER statement enhance the object of database. In structured query language it modifies the properties of database object. The ALTER statement is

- ALTER type of object name of object

RENAME statement is used to rename a database. It's statement is as follows

- RENAME TABLE old name of table to new name of table.

2. Data manipulation language (DML)

It has statements which are used to manage the data within the pattern of objects. Some of thesamples of the statements are as follows:

- SELECT – useful in holding data from a database
- INSERT – helps in inserting data in to a table
- UPDATE – used in updating the data
- DELETE – do the function of deleting the records
- MERGE – this do the UPSERT operation i.e. insert or update operation
- CALL – this calls a structured query language or a java subprogram
- EXPLAIN PLAN – has the parameter of explaining data
- LOCK TABLE – this ha the function of controlling concurrency

These syntax elements are similar to the syntax elements used in computer programming language. Performing the operation of reading of queries is also a component of data manipulation language. Other forms of data manipulation languages (DML) are used by IMS,CODASYL databases.

DML also include the structured query language (SQL) data modifying statements, they modify the saved data but not the pattern of objects. The initial word of the DML statementshas functional capability.

The query statement SELECT is grouped with data statements of structured query language (SQL). In practice there is no such difference and it is viewed to be a portion of DML.

Data manipulation languages contribute to have distinct relishes between database sellers.

They are divided as:

- Procedural programming
- Declarative programming

Initially data manipulation languages were only used in computer programs, but with the coming of structured query languages it is also used in the database executors.

3. **Data Control Language (DCL)**

Data Control Language (DCL) is syntax similar to the programming language, which was used to retrieve the stored or saved data. Examples of the commands in the data control language (DCL) are:

- GRANT – this permits particular users to perform particular tasks
- REVOKE – it blocks the previously granted untrue permissions

The operations which has the authorization of REVOKE are CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

The execution of DCL is transactional; it also has the parameter of rolling back. But the execution of data control language in oracle database does not have the feature of rolling back.

4. **Transaction Control Language (TCL)**

Transaction Control Language (TCL) has commands which are used to manage the transactions or the conduct of a database. They manage the changes made by data manipulation language

statements and also group up the statements in o logical management.

Some examples of it are:

- COMMIT – use to save work
- SAVE POINT – helps in identifying a point in the transaction, can be rolled back tothe identified point
- ROLL BACK – has the feature of restoring the database to the genuine point, sincefrom the last COMMIT
- SET TRANSACTION – have parameter of changing settings like isolation level androll back point

COMMIT command permanently save the transaction in to database.

- It's syntax is: Commit;

ROLL BACK command uses the save point command to jump to save point in transaction.

- It' s syntax is: rollback to name-save point;

SAVE POINT command is used to save a transaction temporarily.

- It's syntax is: Save point name-save point;

Database Users

Database administrators – DBA is responsible for authorizing access to the database, for coordinating and monitoring its use, and acquiring software and hardware resources as needed.

Database designers – identify data to be stored in the database and choosing appropriate structures to represent and store the data. Most of these functions are done before the database is implemented and populated with the data. It is the responsibility of the database designers

to communicate with all prospective users to understand their requirements and come up with a design that meets these requirements. Database designers interact with all potential users and develop views of the database that meet the data and processing requirements of these groups. The final database must support the requirements of all user groups.

End Users

- **Casual End Users** – occasionally access, may need different information each time. Usequery language to specify requests.
- **Naïve or parametric end users** – main job is to query and update the database using standard queries and updates. These canned transactions have been carefully programmed and tested. Examples?
- **Sophisticated end users** – engineers, scientists, analysts who implement applications to meet their requirements.
- **Stand alone users** – maintain personal databases using ready made packages.

DBA

A database administrator's (DBA) primary job is to ensure that data is available, protected from loss and corruption, and easily accessible as needed. Below are some of the chief responsibilities that make up the day-to-day work of a DBA. DSP deliver an outsourced DBAservice in the UK, providing Oracle Support and SQL Server Support; whilst mindset and toolset may be different, whether a database resides on-premise or in a Public / Private Cloud,the role of the DBA is not that different.

1. Software installation and Maintenance

A DBA often collaborates on the initial installation and configuration of a new Oracle, SQL Server etc database. The system administrator sets up hardware and deploys the operating system for the database server,

then the DBA installs the database software and configures it for use. As updates and patches are required, the DBA handles this on-going maintenance. And if a new server is needed, the DBA handles the transfer of data from the existing system to the new platform.

2. Data Extraction, Transformation, and Loading

Known as ETL, data extraction, transformation, and loading refers to efficiently importing large volumes of data that have been extracted from multiple systems into a data warehouse environment. This external data is cleaned up and transformed to fit the desired format so that it can be imported into a central repository.

3. Specialized Data Handling

Today's databases can be massive and may contain unstructured data types such as images, documents, or sound and video files. Managing a very large database (VLDB) may require higher-level skills and additional monitoring and tuning to maintain efficiency.

4. Database Backup and Recovery

DBAs create backup and recovery plans and procedures based on industry best practices, then make sure that the necessary steps are followed. Backups cost time and money, so the DBA may have to persuade management to take necessary precautions to preserve data.

System admins or other personnel may actually create the backups, but it is the DBA's responsibility to make sure that everything is done on schedule.

In the case of a server failure or other form of data loss, the DBA will use existing backups to restore lost information to the system. Different types of failures may require different recovery strategies, and the DBA must be prepared for any eventuality. With technology change, it is becoming ever more typical for a DBA to backup databases to the cloud, Oracle Cloud for Oracle Databases and MS Azure for SQL Server.

5. Security

A DBA needs to know potential weaknesses of the database software and the company's overall system and work to minimize risks. No system is one hundred per cent immune to attacks, but implementing best practices can minimize risks.In the case of a security breach or irregularity, the DBA can consult audit logs to see who has done what to the data. Audit trailsare also important when working with regulated data.

6. Authentication

Setting up employee access is an important aspect of database security. DBAs control who has access and what type of access they are allowed. For instance, a user may have permission to see only certain pieces of information, or they may be denied the ability to make changes to the system.

7. Capacity Planning

The DBA needs to know how large the database currently is and how fast it is growing in order to make predictions about future needs. Storage refers to how much room the database takes up in server and backup space. Capacity refers to usage level. If the company is growing quickly and adding many new users, the DBA will have to create the capacity to handle the extra workload.

8. Performance Monitoring

Monitoring databases for performance issues is part of the on-going system maintenance a DBA performs. If some part of the system is slowing down processing, the DBA may need tomake configuration changes to the software or add additional hardware capacity. Many types of monitoring tools are available, and part of the DBA's job is to understand what they need to track to improve the system. 3rd party organizations can be ideal for outsourcing this aspect, but make sure they offer modern DBA support.

9. Database Tuning

Performance monitoring shows where the database should be tweaked to operate as efficiently as possible. The physical configuration, the way the database is indexed, and how queries are handled can all have a dramatic effect on database performance. With effective monitoring, it is possible to proactively tune a system based on application and usage instead of waiting until a problem develops.

10. Troubleshooting

DBAs are on call for troubleshooting in case of any problems. Whether they need to quickly restore lost data or correct an issue to minimise damage, a DBA needs to quickly understand and respond to problems when they occur.

Introduction to relational model:

The relational model was introduced by Dr.E.F.Codd in 1970.The relational model represents data in the form of two dimensional tables. The organization of data into relational tables is known as the logical view of the database.

Characteristics of Relational Model:

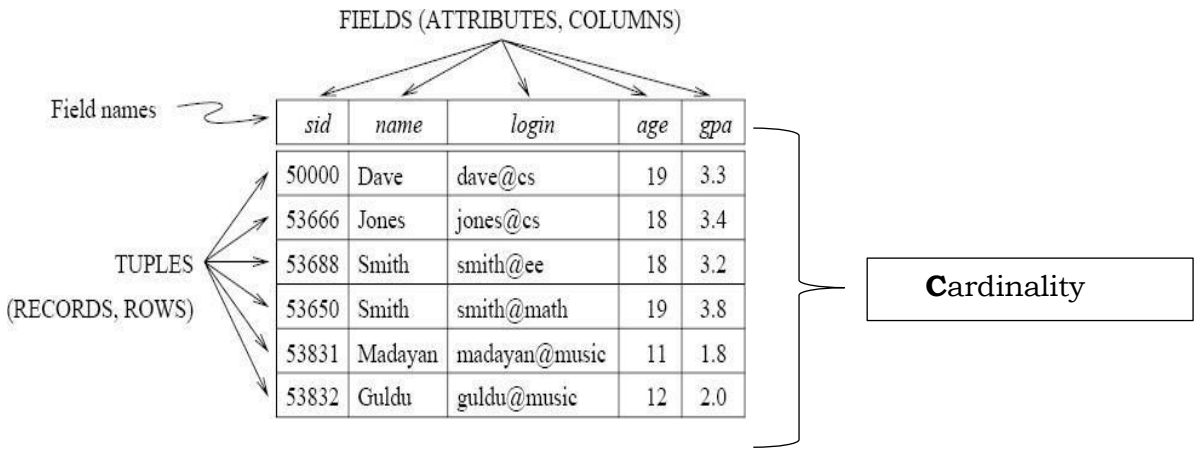
- A relational table eliminates all parent child relationships or instead represented all data in the database as sample row/column tables of data values
- A relation as similar to a table with rows/columns of datavalues
- Each table as an independent entry and there as no physical relationships betweentables
- Relational model of data management is based on set theory
- The user interface used with relational models as non procedural because only what needs to be done as specified and not how it has to be done

Fundamental concepts of relations:

Relation:- A Relation can be thought of as a set of records in the form of two-dimensional table containing rows and columns of data

A relation consists of two things: a relation schema and instance relation.

Relation schema: The relation schema contains the basic information of a table. This information includes the name of the table, the names of the columns and the data types associated with each column



Relation instance: An instance of a relation is a set of types in which each tuple has the same no. of fields as the relation schema

Relational database schema: A relational database schema is a collection of relation schemas, describing one or more relations

Relation cardinality: The Relation cardinality is the no. of tuples in the relation**Relation degree:** The Relation degree as the no.of columns in the relation **Tuples/records:** The rows of the table as also known as records or tuples **Field/attributes:** The columns of the table as also known as fields/attributes

Tabular Representation of Various ER Schemas

The relational model is today the primary data model for commercial data processing applications. It attained its primary position because of its simplicity, which eases the job of the programmer, compared to earlier data models such as the network model or the hierarchical model.

Structure of Relational Databases:

A relational database consists of a collection of **tables**, each of which is assigned a unique name. For example, consider the *instructor* table of Figure:1.5, which stores information about instructors. The table has four column headers: *ID*, *name*, *dept name*, and *salary*. Each row of this table records information about an instructor, consisting of the instructor's *ID*, *name*, *dept name*, and *salary*.

Database Schema

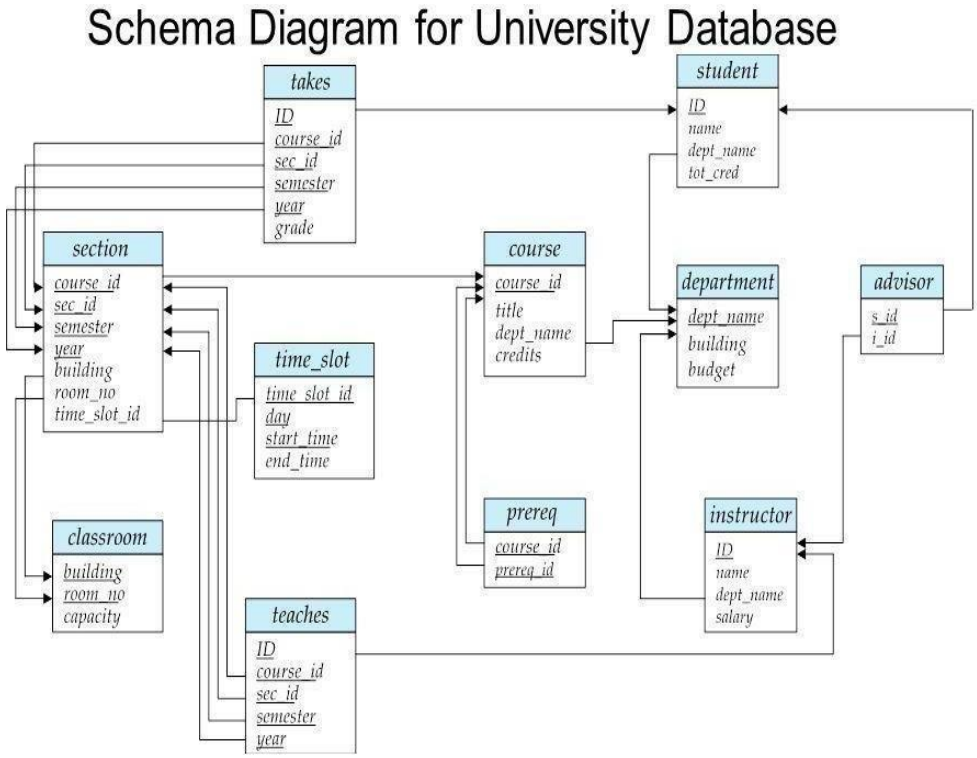
When we talk about a database, we must differentiate between the **database schema**, which is the logical design of the database, and the **database instance**, which is a snapshot of the data in the database at a given instant

in time. The concept of a relation corresponds to the programming- language notion of a variable, while the concept of a **relation schema** corresponds to the programming-language notion of type definition.

Schema Diagrams

A database schema, along with primary key and foreign key dependencies, can be depicted by

schema diagrams. Figure 1.12 shows the schema diagram for our university organization.



Schema diagram for the university database.

Referential integrity constraints other than foreign key constraints are not shown explicitly in schema diagrams.We will study a different diagrammatic representation called the entity- relationship diagram.

ER Diagram Notations

An E-R diagram consists of the following major components:

• Rectangles divided into two parts represent entity sets. The first part, which in this textbook is shaded blue, contains the name of the entity set. The second part contains the names of all the attributes of the entity set.

• Diamonds represent relationship sets.

• Undivided rectangles represent the attributes of a relationship set. Attributes that are part of the primary key are underlined.

• Lines link entity sets to relationship sets.

• Dashed lines link attributes of a relationship set to the relationship set.

• Double lines indicate total participation of an entity in a relationshipset.

• Double diamonds represent identifying relationship sets linked to weak entitysets

Weak Entity Set-

Consider a section entity, which is uniquely identified by a course identifier, semester, year, and section identifier. Clearly, section entities are

related to course entities. Suppose we create a relationship set sec course between entity sets section and course. Now, observe that the information in sec course is redundant, since section already has an attribute course id, which identifies the course with which the section is related. One option to deal with this redundancy is to get rid of the relationship sec course; however, by doing so the relationship between section and course becomes implicit in an attribute, which is not desirable.

The notion of weak entity set formalizes the above intuition. An entity set that does not have sufficient attributes to form a primary key is termed a weak entity set. An entity set that has a primary key is termed a strong entity set.

For a weak entity set to be meaningful, it must be associated with another entity set, called the identifying or owner entity set. Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be existence dependent on the identifying entity set. The identifying entity set is said to own the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the identifying relationship

had a primary key. However, conceptually, a section is still dependent on a course for its existence, which is made explicit by making it a weak entity set.

In E-R diagrams, a weak entity set is depicted via a rectangle, like a strong entity set, but there are two main differences:

- The discriminator of a weak entity is underlined with a dashed, rather than a solid, line.

- The relationship set connecting the weak entity set to the identifying strong entity set is depicted by a double diamond.

Views

Introduction to views

- The dynamic result of one or more relational operations operating on the base relations to produce another relation is called view. A view is a virtual relation that does not necessarily exist in the database. But can be produced upon request by a particular user at the time of request.
- A view is object that gives the user a logical view of data from an underlying table or tables. You can restrict what users can view by allowing them to see only a few columns from a table.

Purpose of views

- The view mechanism is desirable for several reasons.
- It simplifies queries.
- It can be queried as a base table.
- It provides a powerful and flexible security mechanism by hiding parts of the database from certain users.
- It permits users to access data in a way that is customized to their needs, so that the same data can be seen by different users in different ways at the same time.

Updating views

- All updates to a base relation should be immediately reflected in all views that a single base relation and containing either the primary key or a candidate key of the baserelation.
- Updates are not allowed through views involving multiple base relations.
- Updates are not allowed through views involving aggregation or grouping operations.

Creating views Syntax:

```
CREATE VIEW  viewname as SELECT

columnname, cloumnname

FROM  tablename

 WHERE columnname=expression list;
```

Examples: create view on book table which contains two fields title, and author name.

SQL> create view V_book as select title, author_name from book;

View created.

SQL>select * from V_book;

Output:

Title	**Author_name**
Oracle	Arora
DBMS	Basu
DOS	Sinha
ADBMS	Basu
Unix	Kapoor

Selecting Data from a view

Example: Display all the title of book written by author 'Basu'.

SQL> select title from V_Book Where
author_name='Basu';

Output:

Title
DBMS
ADBMS

Updatable Views

Views can also be used for data manipulation i.e., the user can perform Insert, Update and the Delete operations on the view. The views on which data manipulation can be done are called Updatable

views, views that do not allow data manipulation are called Read only Views. When you give a view name in the update, insert or delete statement, the modification to the data will be passed to the underlying table.

For the view to be updatable, it should meet following criteria:

- The view must be created on a single table.
- The primary key column of the table should be included in the view.
- Aggregate functions cannot be used in the select statement.
- The select statement used for creating a view should not included Distinct, Group by or Having clause.
- The select statement used for creating a view should not include sub queries.
- It must not use constant, string or values expression like total/5.

Destroying/Altering Tables and Views Altering Table

The definition of the table is changed using ALTER TABLE statement. The ALTER TABLE is used to add, delete or modify columns in an existing table explained below:

1) **ALTER TABLE.....ADD.....**

   This is used to add some extra columns into an existing table. The generalized format is given below.

```
ALTER TABLE relation_name ADD(new
field1 datatype(size), new field2
datatype(size)'.....
```

Example:

ADD customer phone and fax number in the customer relation.

SQL> ALTER TABLE Customer ADD(cust_ph_no varchar(15),

cust_fax_no varchar(15));

SQL>Table created.

2) **ALTER TABLE .....MODIFY**

This form is used to change the width as well as data type of existing relations. the generalized syntax of this from is shown below.

ALTER TABLE relation_name MODIFY($field_1$ new data type(size), $field_2$ new data type(size), $field_n$ new data type(size));

- **Example:**

Modify the data type of the publication year as numeric data type.

SQL> ALTER TABLE Book MODIFY(pub_year number(4));

SQL>Table created.

Restrictions of the Alter Table

Using the alter table clause you perform the following tasks:

Change the name of the table.
Change the name of the
column.

Drop a column.

Decrease the size of a column if table data exists.

3) **ALETR TABLE……DELETE**

To delete a column in a table , use the following syntax:

```
ALTER TABLE table_name DROP
COLUMN column_name
```

- **Example**: Drop customer fax number

from the customer table.

SQL> ALTER TABLE customer

DROP COLUMN cust_fax_no;

DELETING TABLE

The tables are deleted permanently from the database using DROP TABLE command. We remove all the data from the table using

TRUNCATE TABLE command. It is explained below:

1) **DROP TABLE**

This command is used to delete a table. The generalized syntax if this form is given below:

```
DROP TABLE relation_name
```

- **Example**: write the command for deleting special_customer relation.

  SQL DROP TABLE Special_customer;

  TABLR dropped.

2) **Truncate a table**

Truncating a table is removing all records from the table. The structure of the table stays intact. The SQL language has a DELETE statement which can be used to remove one or more (or all) rows from a table. Truncation releases storage space occupied by the table, but deletion does not.

The **syntax:**

```
TRUNCATE TABLE table_name;
```

- **Example:**

SQL> TRUNCATE TABLE student;

Deleting view

A view can be dropped by using the DROP VIEW command.

Syntax:

```
DROP VIEW viewname;
```

- **Example:**

DROP VIEW V_Book;

Triggers.

A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an active database. A trigger description contains three parts:

Event: A change to the database that activates the trigger.

Condition: A query or test that is run when the trigger is activated.

Action: A procedure that is executed when the trigger is activated and its con-dition is true.

A trigger *action* can examine the answers to the query in the condition part of the trigger, refer to old and new values of tuples modified by the statement activating the trigger, execute new queries, and make changes to the database.

Examples of Triggers in SQL

The examples shown in Figure 5.19, written using Oracle 7 Server syntax for defining triggers, illustrate the basic concepts behind triggers. (The SQL:1999 syntax for these triggers is similar; we will see an example using SQL:1999 syntax shortly.) The trigger called *init count* initializes a counter variable before every execution of an INSERT statement that adds tuples to the Students relation. The trigger called *incr count* increments the counter for each inserted tuple that satisfies the condition *age* < 18.

```
CREATE TRIGGER init count BEFORE INSERT
   ON Students /* Event */ DECLARE
      count INTEGER;
   BEGIN /*action*/
         Count:=0;

   END

CREATE TRIGGER incr count AFTER INSERT ON Students /* Event
   */ WHEN (new.age < 18) /* Condition; 'new' is
   just-inserted tuple */ FOR EACH ROW
   BEGIN       /* Action; a procedure in Oracle's
      PL/SQL syntax */
   cunt := count + 1; END
```

(identifying the modified table, Students, and the kind of modifying statement, an INSERT), and the third field is the number of inserted Students tuples with *age* < 18. (The trigger in Figure 5.19 only computes the count; an additional trigger is required to insert the appropriate tuple into the statistics table.)

```
CREATE TRIGGER set count AFTER INSERT ON Students
 /* Event */
REFERENCING NEW TABLE AS Inserted Tuples
FOR EACH STATEMENT
INSERT        /* Action */
INTO StatisticsTable(ModifiedTable, ModificationType, Count)
SELECT
'Students', 'Insert',COUNT * FROM
InsertedTuples I WHEREI.age < 18
BEGIN
  - count := 0;
END
```

SQL: Overview, The Form of Basic SQL Query -UNION, INTERSECT, and EXCEPT– join operations: equi join and non equi join-Nested queries - correlated and uncorrelated- Aggregate Functions-Null values, GROUPBY- HAVING Clause.

THE FORM OF A BASIC SQL QUERY

This section presents the syntax of a simple SQL query and explains its meaning through a *conceptual evaluation strategy*. A conceptual evaluation strategy is a way to evaluate the query that is intended to be easy to understand, rather than

efficient. A DBMS would typically execute a query in a different and more efficient way.

Sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1An Instance *S* 3 of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 5.2 An Instance *R*2 of Reserves

bid	*bname*	*color*
101	*Interlake*	*blue*
102	*Interlake*	*red*
103	*Clipper*	*green*
104	*Marine*	*red*

(Q15) Find the names and ages of all sailors.

```
SELECT DISTINCT S.sname, S.age FROM Sailors S
```

The answer to this query with and without the keyword DISTINCT on instance *S*3 of Sailors is shown in Figures 5.4 and 5.5. The only difference is that the tuple for Horatio appears twice if DISTINCT is omitted; this is because there are two sailors called Horatio and age 35.

(Q11) Find all sailors with a rating above 7.

SELECT S.sid, S.sname, S.rating, S.age FROM Sailors AS S WHERE S.rating > 7

(Q16) Find the sids of sailors who have reserved a red boat.

SELECT R.sid FROM Boats B, Reserves R WHERE B.bid = R.bid AND B.color = 'red'

(Q2) Find the names of sailors who have reserved a red boat.

SELECT S.sname FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'

(Q3) Find the colors of boats reserved by Lubber.

SELECT B.color FROM Sailors S, Reserves R, Boats B WHERE S.sid = R.sid AND R.bid = B.bid AND S.sname = 'Lubber'

(Q4) Find the names of sailors who have reserved at least one boat.

SELECT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid

Expressions and Strings in the SELECT Command

SQL supports a more general version of the select-list than just a list of columns. Each item in a select-list can be of the form expression AS column name, where expression is any arithmetic or string expression over column names (possibly prefixed by range variables) and constants.

(Q5) Compute increments for the ratings of persons who have sailed two differentboats on the same day.

```
SELECT S.sname, S.rating+1 AS rating FROM Sailors
S, Reserves R1, Reserves R2 WHERE S.sid = R1.sid
AND S.sid = R2.sid AND R1.day = R2.day AND R1.bid
<> R2.bid
```

Also, each item in a *qualification* can be as general as *expression1* = *expression2*.

```
SELECT S1.sname AS name1, S2.sname AS name2
FROM Sailors S1,Sailors S2 WHERE 2*S1.rating =
S2.rating-1.
```

(Q6) Find the ages of sailors whose name begins and ends with B and has at leastthree characters.

SELECT S.age FROM Sailors S WHERE S.sname LIKE 'B %B'

The only such sailor is Bob, and his age is 63.5.

UNION, INTERSECT, AND EXCEPT

SQL provides three set-manipulation constructs that extend the basic query form pre- sented earlier. Since the answer to a query is a multiset of rows, it is natural to consider the use of operations such as union, intersection, and difference. SQL supports these operations under the names UNION, INTERSECT, and EXCEPT.[4] SQL also provides other set operations: IN (to

check if an element is in a given set),op ANY,op ALL(tocom-pare a value with the elements in a given set, using comparison operator op), and EXISTS (to check if a set is empty). IN and EXISTS can be prefixed by NOT,

with the obvious modification to their meaning. We cover UNION, INTERSECT, and EXCEPT in this section. Consider the following query:

(Q1) Find the names of sailors who have reserved both a red and a green boat.

SELECT S.sname FROM Sailors S, Reserves R1, Boats B1, Reserves R2, Boats B2 WHERE S.sid = R1.sid AND R1.bid = B1.bid AND S.sid = R2.sid AND R2.bid

= B2.bid AND B1.color='red' AND B2.color = 'green'

(Q2) Find the sids *of all sailors who have reserved red boats but not green boats.*

SELECT S.sid FROM Sailors S, Reserves R, Boats B

WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red' EXCEPT SELECT S2.sid FROM Sailors S2, Reserves R2, Boats B2WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'

Joins

The *join* operation is one of the most useful operations in relational algebra and is the most commonly used way to combine information from two or more relations. Althougha join can be defined as a cross-product followed by selections and projections, joins arise much more frequently in practice than plain cross-products.joins have received a lot of attention, and there are several variants of the join operation.

Condition Joins

The most general version of the join operation accepts a *join condition c* and a pair of relation instances as

arguments, and returns a relation instance. The *join condition* is identical to a *selection condition* in form. The operation is defined as follows:

$R \text{ a}\rhd_c S = \sigma_c(R \times S)$

Thus a⊳ is defined to be a cross-product followed by a selection. Note that the condition *c* can (and typically *does*) refer to attributes of both *R* and *S*.

(*sid*)	*sname*	*rating*	*age*	(*sid*)	*bid*	*day*
22	*Dustin*	*7*	*45.0*	58	*103*	*11/12/96*
31	*Lubber*	*8*	*55.5*	58	*103*	*11/12/96*

Figure 4.12 *S*1 a⊳*S*1.*sid*<*R*1.*sid R*1

Equijoin

A common special case of the join operation *R* a⊳ *S* is when the *join condition* con- sists solely of equalities (connected by A) of the form *R.name*1 = *S.name*2, that is, equalities between two fields in *R* and *S*. In this case, obviously, there is some redun- dancy in retaining both attributes in the result.

Natural Join

A further special case of the join operation *R* a⊳ *S* is an equijoin in which equalities are specified on *all* fields having the same name in *R* and *S*. In this case, we can simply omit the join condition; the default is that the join condition is a collection of equalities on all common fields.

Non Equi Join

The SQL NON EQUI JOIN uses comparison operator instead of the equal sign like **>, <, >=,**
<= along with conditions.

```
SELECT *
FROM table_name1, table_name2
WHERE table_name1.column [> | < | >= | <= ] table_name2.column;
```

NESTED QUERIES

A nested query is a querythat has another query embedded within it; the embeddedquery is called a subquery.

(Q1) Find the names of sailors who have reserved boat 103.

```
SELECT S.snameFROM      Sailors S
WHERE S.sid IN ( SELECT R.sid
                 FROM    Reserves R
                 WHERE R.bid = 103 )
```

(Q2) Find the names of sailors who have reserved a red boat.

```
SELECT          S.snameFROM      Sailors S
WHERE S.sid IN ( SELECT R.sid
                 FROM    Reserves R
                 WHERE R.bid IN ( SELECT B.bid
  FROM    Boats B
          WHERE B.color = 'red' )
```

(Q3) Find the names of sailors who have *not* reserved a red boat**.**

```
SELECT S.snameFROMSailors S
WHERE S.sid NOT IN ( SELECT R.sid
                     FROM    Reserves R
                     WHERE  R.bid IN ( SELECT B.bid
                                       FROM    Boats B
                                       WHERE B.color = 'red' )
```

Correlated Nested Queries

In the nested queries that we have seen thus far, the inner subquery has been completelyindependent of the outer query:

(Q1) Find the names of sailors who have reserved boat number 103.

```
SELECT
S.sname
FROM

Sailors
S
WHERE
EXISTS
(
SELECT
*
                    FROM    Reserves R
WHERE  R.bid = 103
AND R.sid = S.sid )
```

Set-Comparison Operators

(Q1) Find sailors whose rating is better than some sailor called Horatio.

```
SELECT S.sid
FROM    Sailors S
WHERE S.rating > ANY ( SELECT S2.rating
FROM    Sailors S2
                    WHERE S2.sname = 'Horatio' )
```

(Q2) Find the sailors with the highest rating .

```
SELECT S.sid
FROM Sailors S
WHERE       S.rating >= ALL (  SELECT
S2.rating FROM Sailors S2 )
```

More Examples of Nested Queries

(Q1) Find the names of sailors who have reserved both a red and a green boat.

```
SELECT S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid
        AND B.color = 'red'AND S.sid IN (
        SELECT S2.sid
                FROM    Sailors S2, Boats
                B2, Reserves R2 WHERE
                S2.sid = R2.sid AND
                R2.bid = B2.bid
AND B2.color = 'green' )
```

Noncorrelated

There are two kind of subquery in SQL one is called non-correlated and other is called correlated subquery. In non correlated subquery, **inner query doesn't depend on outer query** and can run as stand alone query.Subquery used along-with IN or NOT IN sql clause is good examples of Noncorrelated subquery in SQL. Let's a **noncorrelated subquery example** to understand it better

NonCorrelated subquery are used along-with IN and NOT IN clause. here is an example ofsubquery with IN clause in SQL.

SQL query: Find all stocks from United States and India

```
mysql> SELECT COMPANY FROM Stock WHERE LISTED_ON_EXCHANGE IN (SELECT RIC FRO
M Market WHERE COUNTRY='United States' OR COUNTRY= 'INDIA');
+-----------------------------+
| COMPANY                     |
+-----------------------------+
| Google Inc                  |
| Goldman Sachs GROUP Inc     |
| InfoSys                     |
+-----------------------------+
```

AGGREGATE OPERATORS

We now consider a powerful class of constructs for computing *aggregate values* such asMIN

and SUM.

1. COUNT ([DISTINCT] A): The number of (unique) values in the A column.
2. SUM ([DISTINCT] A): The sum of all (unique) values in the A column.
3. AVG ([DISTINCT] A): The average of all (unique) values in the A column.
4. MAX (A): The maximum value in the A column.
5. MIN (A): The minimum value in the A column.

(Q1) Find the average age of all sailors.

```
SELECT AVG (S.age) FROMSailors S
```

Q2) Find the average age of sailors with a rating of 10.

```
SELECT AVG (S.age)
FROM    Sailors S
WHERE S.rating = 10
```

```
SELECT  S.sname, MAX  (S.age)
FROM  Sailors S
```

Q3) Count the number of sailors.

```
SELECT COUNT (*)
FROM  Sailors S
```

NULL VALUES

we have assumed that column values in a row are always known. In practice column values can be unknown. For example, when a sailor, say Dan, joins a yacht club, he may not yet have a rating assigned. Since the definition for the Sailors table has a *rating* column, what row should we insert for Dan? What is needed here is a special value that denotes *unknown*.

SQL provides a special column value called *null* to use in such situations. We use *null* when the column value is either *unknown* or *inapplicable*. Using our Sailor table definition, we might enter the row 〈 98*, Dan, null,* 39 〉 to represent Dan. The presence of *null* values complicates many issues, and we consider the impact of *null* values on SQL in this section.

Comparisons Using Null Values

Consider a comparison such as *rating* = *8*. If this is applied to the row for Dan, is this condition true or false? Since Dan's rating is unknown, it is reasonable to say that this comparison should evaluate to the value unknown.

SQL also provides a special comparison operator IS NULL to test whether a column valueis *null*; for example, we can say *rating* IS NULL, which would evaluate to true on the row representing Dan.

We can also say *rating* IS NOT NULL, which would evaluate to false onthe row for Dan.

Logical Connectives AND, OR, and NOT

Now, what about boolean expressions such as *rating* = 8 OR *age* < 40 and *rating* = 8 AND *age* < 40? Considering the row for Dan again, because *age* < 40, the first expression evaluates to true regardless of the value of *rating*, but what about the second?We can only say unknown.

The GROUP BY and HAVING Clauses

we want to apply aggregate operations to each of a number of groups of rows in a relation, where the number of groups depends on the relation instance (i.e., is not known in advance). ***(Q31) Find the age of the youngest sailor for each rating level.***

```
SELECT MIN (S.age)
FROM    Sailors S
WHERE S.rating = i
```

Q32) Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 yearsold) for each rating level with at least two such sailors.

```
SELECT     S.rating, MIN (S.age) AS
minageGROUP BY S.ratingHAVING
                COUNT (*) > 1
```

More Examples of Aggregate Queries

Q3) For each red boat, find the number of reservations for this boat.

```
SELECT B.bid, COUNT (*) AS sailorcount FROM
Boats B, Reserves RWHERE R.bid = B.bid AND
B.color = 'red' GROUP BY B.bid
```

```
SELECT B.bid, COUNT (*) AS sailorcount FROM
Boats B, Reserves RWHERE R.bid = B.bid GROUP
BY B.bid HAVING B.color = 'red'
```

(Q4) Find the average age of sailors for each rating level that has at least two sailors.

```
SELECT    S.rating, AVG
(S.age) AS avgageFROM
          Sailors S

GROUP
BY
S.rating
HAVING    COUNT (*) > 1
```

(Q5) Find the average age of sailors who are of voting age (i.e., at least 18 years old) for

each rating level that has at least two sailors.

```
SELECT        S.rating, AVG
( S.age ) AS avgageFROM
          Sailors S

WHERE S. age >= 18

GROUP BY S.rating

HAVING 1 < ( SELECT COUNT (*)


                FROM Sailors S2 WHERE S.rating = S2.rating
```

(Q6) Find the average age of sailors who are of voting age (i.e., at least 18 yearsold) for each rating level that has at least two *such* sailors.

```
SELECT        S.rating, AVG
( S.age ) AS avgageFROM
          Sailors S

WHERE S. age > 18

GROUP BY S.rating


HAVING 1 < ( SELECT COUNT (*)

                FROM Sailors S2

                WHERE S.rating = S2.rating AND S2.age >= 18 )
```

The above formulation of the query reflects the fact that it is a variant of Q35. The answer to Q36 on instance *S*3 is shown in Figure 5.16. It differs from the answer to Q35 in that there is no tuple for rating 10, since there is only one tuple with rating 10 and *age*≥ 18.

```
SELECT        S.rating, AVG ( S.age ) AS
avgageFROM          Sailors S

WHERE S. age > 18

GROUP BY S.rating HAVING
            COUNT (*) > 1
```

This formulation of Q36 takes advantage of the fact that the WHERE clause is applied before grouping is done; thus, only sailors with *age* > 18 are left when grouping is done.It is instructive to consider yet another way of writing this query:

```
SELECT Temp.rating, Temp.avgage

FROM ( SELECT S.rating, AVG ( S.age ) AS

avgage, COUNT (*) ASratingcount

        FROM Sailors S WHERE S. age > 18 GROUP BY
S.rating ) AS TempWHERE Temp.ratingcount > 1
```

Parallelism in Query in DBMS

Parallelism in a query allows us to parallel execution of multiple queries by decomposing them into the parts that work in parallel. This can be achieved by shared-nothing architecture. Parallelism is also used in fastening the process of a query execution as more and more resources like processors and disks are provided. We can achieve parallelism in a query by the following methods :

1. I/O parallelism
2. Intra-query parallelism
3. Inter-query parallelism
4. Intra-operation parallelism
5. Inter-operation parallelism

1. I/O parallelism :

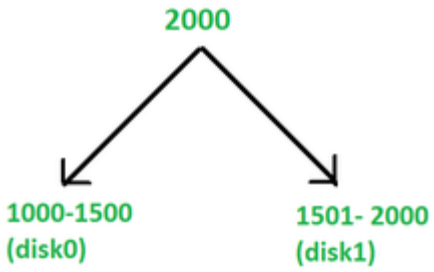
It is a form of parallelism in which the relations are partitioned on multiple disks a motive to reduce the retrieval time of relations from the disk. Within, the data inputted is partitioned and then processing is done in parallel with each partition. The results are merged after processing all the partitioned data. It is also known as **data-partitioning**. Hash partitioning has the advantage that it provides an even distribution of data across the disks and it is also best suited for those point queries that are based on the partitioning attribute. It is to be noted that partitioning is useful for the sequential scans of the entire table placed on '*n*' number of disks and the time taken to scan the relationship is approximately *1/n* of the time required to scan the table on a single disk system. We have four types of partitioning in I/O parallelism:

- **Hash partitioning**

  As we already know, a Hash Function is a fast, mathematical function. Each row of the original relationship is hashed on partitioning attributes. For example, let's assume that there are 4 disks *disk1, disk2, disk3,* and *disk4* through which the data is to be partitioned. Now if the Function returns 3, then the row is placed on *disk3*.

- **Range partitioning**

  In range partitioning, it issues continuous attribute value ranges to each disk. For example, we have 3 disks numbered 0, 1, and 2 in range partitioning, and may assign relation with a value that is less than 5 to disk0, values between 5-40 to disk1, and values that are greater than 40 to disk2. It has some advantages, like it involves placing shuffles containing attribute values that fall within a certain range on the disk. See figure 1: *Range partitioning given* below:



- **Round-robin partitioning**

  In Round Robin partitioning, the relations are studied in any order. The ith tuple is sent to the disk number(i % n). So, disks take turns receiving new rows of data. This technique ensures the even distribution of tuples across disks and is ideally suitable for applications that wish to read the entire relation sequentially for each query.

- **Schema partitioning**

  In schema partitioning, different tables within a database are placed on different disks. See figure 2 below:

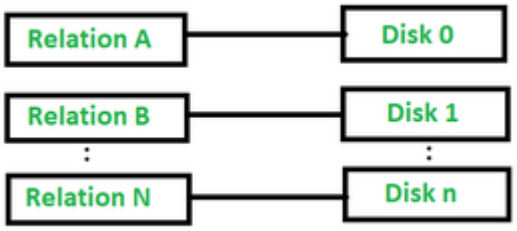


figure – 2

2. Intra-query parallelism :

Intra-query parallelism refers to the execution of a single query in a parallel process on different CPUs using a shared-nothing paralleling architecture technique. This uses two types of approaches:

- **First approach**

  In this approach, each CPU can execute the duplicate task against some data portion.
- **Second approach**

  In this approach, the task can be divided into different sectors with each CPU executing a distinct subtask.

2. Inter-query parallelism :

In Inter-query parallelism, there is an execution of multiple transactions by each CPU. It is called parallel transaction processing. DBMS uses transaction dispatching to carry inter query parallelism. We can also use some different methods, like efficient lock management. In this method, each query is run sequentially, which leads to slowing down the running of long queries. In such cases, DBMS must understand the locks held by different transactions running on different processes. Inter query parallelism on shared disk architecture performs best when transactions that execute in parallel do not accept the same data. Also, it is the easiest form of parallelism in DBMS, and there is an increased transaction throughput.

3. Intra-operation parallelism :

Intra-operation parallelism is a sort of parallelism in which we parallelize the execution of each individual operation of a task like sorting, joins, projections, and so on. The level of parallelism is very high in intra-operation parallelism. This type of parallelism is natural in database systems. Let's take an SQL query example:

*SELECT * FROM Vehicles ORDER BY Model_Number;*

In the above query, the relational operation is sorting and since a relation can have a large number of records in it, the operation can be performed on different subsets of the relation in multiple processors, which reduces the time required to sort the data.

5. Inter-operation parallelism :

When different operations in a query expression are executed in parallel, then it is called inter-operation parallelism. They are of two types –

- **Pipelined parallelism –**

  In pipeline parallelism, the output row of one operation is consumed by the second operation even before the first operation has produced the entire set of rows in its output. Also, it is possible to run these two operations simultaneously on different CPUs, so that one operation consumes tuples in parallel with another operation, reducing them. It is useful for the small number of CPUs and avoids writing of intermediate results to disk.

- **Independent parallelism –**

  In this parallelism, the operations in query expressions that are not dependent on each other can be executed in parallel. This parallelism is very useful in the case of the lower degree of parallelism.

UNIT-II

Distributed Databases

- Heterogeneous and Homogeneous Databases
- Distributed Data Storage Distributed Transactions Commit Protocols
- Concurrency Control in Distributed Databases
- Availability
- Distributed Query Processing
- Heterogeneous
- Distributed Databases
- Directory Systems

Distributed Database System

- A distributed database system consists of loosely coupled sites that share no physical component
- Database systems that run on each site are independent of each other
- Transactions may access data at one or more sites

Homogeneous Distributed Databases

- ✓ In a homogeneous distributed Database
- ✓ All sites have identical software

- ✓ Are aware of each other and agree to cooperate in processing user requests.
- ✓ Each site surrenders part of its autonomy in terms of right to change schemas or software
- ✓ Appears to user as a single system
- ✓ In a heterogeneous distributed database
- ✓ Different sites may use different schemas and software
- ✓ Difference in schema is a major problem for query processing

Difference in software is a major problem for transaction

Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing

Distributed Data Storage

- Assume relational data model Replication
  - ✓ System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.

- Fragmentation
  - ✓ Relation is partitioned into several fragments stored in distinct sites Replication and fragmentation can be combined

- ✓ Relation is partitioned into several fragments: system maintains several identical replicas of each such fragment.

Data Replication

- A relation or fragment of a relation is **replicated** if it is stored redundantly in two or more sites.
- Full replication of a relation is the case where the relation is stored at all sites.
- Fully redundant databases are those in which every site contains a copy of the entire database.

Advantages of Replication

- **Availability**: failure of site containing relation *r* does not result in unavailability of *r* is replicas exist.
- **Parallelism**: queries on *r* may be processed by several nodes in parallel.
- **Reduced data transfer**: relation *r* is available locally at each site containing a replica of *r*.

Disadvantages of Replication

- Increased cost of updates: each replica of relation *r* must be updated.
- Increased complexity of concurrency control: concurrent updates to distinct replicas may lead to inconsistent data

unless special concurrency control mechanisms are implemented.

- One solution: choose one copy as **primary copy** and apply concurrency control operations on primary copy

Data Fragmentation

Division of relation r into fragments r_1, r_2, ..., r_n which contain sufficient information to reconstruct relation r.

Horizontal fragmentation: each tuple of r is assigned to one or more fragments

Vertical fragmentation: the schema for relation r is split into several smaller schemas

- All schemas must contain a common candidate key (or superkey) to ensure lossless join property.
- A special attribute, the tuple-id attribute may be added to each schema to serve as a candidate key.

HORIZONTAL FRAGMENTATION OF ACCOUNT RELATION

branch_name	account_number	balance
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

$$account_1 = \sigma_{branch\_name="Hillside"}(account)$$

branch_name	account_number	Balance
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

$$account_2 = \sigma_{branch\_name="Valleyview"}(account)$$

VERTICAL FRAGMENTATION OF EMPLOYEE_INFO RELATION

branch_name	*customer_name*	*tuple_id*
Hillside Hillside Valleyview Valleyview Hillside Valleyview Valleyview	Lowman Camp Camp Kahn Kahn Kahn Green	1 2 3 4 5 6 7

$$deposit_1 = \Pi_{branch\_name,\ customer\_name,\ tuple\_id}\ (employee\_info\ )$$

Advantages of Fragmentation

Horizontal:

- allows parallel processing on fragments of a relation

- allows a relation to be split so that tuples are located where they are most frequently accessed

Vertical:

- allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed tuple-id attribute allows efficient joining of vertical fragments allows parallel processing on a relation
- Vertical and horizontal fragmentation can be mixed.

- Fragments may be successively fragmented to an arbitrary depth.

Data Transparency

Data transparency: Degree to which system user may remain unaware of the details of how and where the data items are stored in a distributed system

Consider transparency issues in relation to:

1. Fragmentation transparency
2. Replication transparency
3. Location transparency

Naming of Data Items

1. Every data item must have a system-wide unique name.
2. It should be possible to find the location of data items efficiently.
3. It should be possible to change the location of data items transparently.
4. Each site should be able to create new data items autonomously.

System Failure Modes

Failures unique to distributed systems:

Failure of a site.

Loss of massages

- Handled by network transmission control protocols such as TCP-IP

Failure of a communication link

- Handled by network protocols, by routing messages via alternative links

Network partition

- A network is said to be **partitioned** when it has been split into two or more subsystems that lack any connection between them

Note: a subsystem may consist of a single node Network partitioning and site failures are generally indistinguishable.

Commit Protocols

Commit protocols are used to ensure atomicity across sites a transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.

Two Phase Commit Protocol (2PC)

- ✓ Assumes **fail-stop** model – failed sites simply stop working, and do not cause any other harm, such as sending incorrect messages to other sites.

- ✓ Execution of the protocol is initiated by the coordinator after the last step of the transaction has been reached.

- ✓ The protocol involves all the local sites at which the transaction executed.

- ✓ Let T be a transaction initiated at site S_i, and let the transaction coordinator at S_i be C_i

Handling of Failures

When site S_i recovers, it examines its log to determine the fate of transactions active at the time of the failure.

Log contain <***commit*** *T*> *record:* txn had completed, nothing to be done n Log contains <**abort** *T*> record: txn had completed, nothing to be done

Log contains <ready T> record: site must consult C to determine the fate of *T*.

i

If *T* committed, **redo** (*T*); write <**commit** *T*> record If *T* aborted, **undo** (*T*)

The log contains no log records concerning T:

Implies that S failed before responding to the **prepare** *T* message from C_i since the failure of S_k precludes the sending of such a response, coordinator C_1 must abort *T* S_k must execute **undo** (*T*)

Recovery and Concurrency Control

In-doubt transactions have a <**ready** *T*>, but neither a <**commit** *T*>, nor an <**abort** *T*> log record.

The recovering site must determine the commit-abort status of such transactions by contacting other sites; this can slow and potentially block recovery.

Recovery algorithms can note lock information in the log.

Instead of <**ready** *T*>, write out <**ready** *T*, *L*> *L* = list of locks held by *T* when the log is written (read locks can be omitted).

For every in-doubt transaction *T*, all the locks noted in the <**ready** *T*, *L*> log record are reacquired.

After lock reacquisition, transaction processing can resume; the commit or rollback of in-doubt transactions is performed concurrently with the execution of new transactions.

Three Phase Commit (3PC)

Assumptions:

1. No network partitioning
2. At any point, at least one site must be up.
3. At most K sites (participants as well as coordinator) can fail

Phase 1: Obtaining Preliminary Decision: Identical to 2PC

Phase 1. 1 Every site is ready to commit if instructed to do so

Phase 2 of 2PC is split into 2 phases, Phase 2 and Phase 3 of 3PC

In phase 2 coordinator makes a decision as in 2PC (called the pre-commit decision) and records it in multiple (at least K) sites

In phase 3, coordinator sends commit/abort message to all participating sites, Under 3PC, knowledge of pre-commit decision can be used to commit despite coordinator failure

Avoids blocking problem as long as < K sites fail n

Drawbacks:

Higher overheads assumptions may not be satisfied in practice

Concurrency Control

Modify concurrency control schemes for use in distributed environment.

- We assume that each site participates in the execution of a commit protocol to ensure global transaction atomicity.
- We assume all replicas of any item are updated
- Will see how to relax this in case of site failures later

Time stamping

Timestamp based concurrency-control protocols can be used in distributed systems

Each transaction must be given a unique timestamp

Main problem: how to generate a timestamp in a distributed fashion

- Each site generates a unique local timestamp using either a logical counter or the local clock.

- Global unique timestamp is obtained by concatenating the unique local timestamp with the unique identifier.

Define within each site S_i a **logical clock** (LC_i), which generates the unique local timestamp

Require that S_i advance its logical clock whenever a request is received from a transaction Ti with timestamp < x,y> and x is greater that the current value of LC_i.

In this case, site S_i advances its logical clock to the value $x + 1$.

Distributed Query Processing

For centralized systems, the primary criterion for measuring the cost of a particular strategy is the number of disk accesses.

- In a distributed system, other issues must be taken into account:
- The cost of a data transmission over the network.
- The potential gain in performance from having several sites process parts of the query in parallel.

UNIT-III

SPATIAL DATABASE

A spatial database is a general-purpose database (usually a relational database) that has been enhanced to include spatial data that represents objects defined in a geometric space, along with tools for querying and analysing such data.

Geodatabase:

A Geographic Database, sometimes known as a Geodatabase, is a Georeferenced Spatial Database that is used to store and modify geodata or information about a specific place on Earth. Additionally, the term "geodatabase" can refer to a collection of exclusive geographic database formats called **Geodatabase.**

Characteristics of Spatial Database:

One or more spatial data types that enable the recording of spatial data as values in a table are the fundamental capability that a spatial extension to a database adds. Based on the vector data model, a single spatial value is often a geometric primitive (points, lines, polygon, etc.). The OGC Simple Features definition for describing geometric primitives serves as the foundation for most spatial databases' data types. Raster data can also be stored in some spatial databases. Spatial Databases must support the tracking and manipulation of coordinate systems since every geographic place must be described using a spatial reference system.

The addition of geographic capabilities to the query language (such as SQL), which gives the Spatial Database access to the exact query, analysis, and

manipulation operations as standard GIS software, is the second significant functionality extension in a spatial database. This feature is implemented as a collection of new methods that can be used in SQL SELECT statements in the majority of Relational Database Management Systems.

There are Several Types of Operations like:

- **Measurement:**
  Computes geometry distance, polygon area, line length, etc.
- **Geoprocessing:**
  Create new features by changing existing ones, for as by surrounding them with a buffer or by intersecting features.
- **Geometry Constructors:**
  Specifies the vertices (points or nodes) that define the form to create new geometries.
- **Observer Functions:**
  Queries that give detailed answers on a feature, like the location of a circle's center.
- **Predicates:**
  True/false questions about the spatial relationships between geometries are permissible.

Spatial Index:

Indexes are frequently used in database systems to provide faster and more effective data access and search. But spatial queries are not a good fit for this index. Instead, to improve database efficiency, geographical databases employ something similar to a distinct index known as a Spatial Index. A system must be able to obtain data from a vast collection of items without actually searching them all. Hence Spatial Indexing is crucial. In addition to filtering, it ought to better allow connections between objects from various classes.

In addition to indexes, geographical databases also provide spatial data types in their query language and data model. To give a basic abstraction and represent the structure of the spatial figures with their related interactions and processes in the geographical environment, these databases require unique sorts of data types. The system would be unable to provide the level of modelling that a spatial database enables without these kinds of data types.

Spatial Query:

A unique kind of sql query supported by spatial databases, especially geodatabases, is known as a Spatial Query. The queries have a number of significant differences from non-spatial SQL queries. The usage of geometry data types, including points, lines, and polygons, as well as the fact that these queries take the spatial relationship between these geometries into account, are two of the most crucial features.

The Predicate Calculus

A predicate is an expression of one or more variables defined on some specific domain. A predicate with variables can be made a proposition by either assigning a value to the variable or by quantifying the variable.

Consider the following statement.

- Ram is a student.

Now consider the above statement in terms of Predicate calculus.

- Here "is a student" is a predicate and Ram is subject.
- Let's denote "Ram" as x and "is a student" as a predicate P then we can write the above statement as P(x).
- Generally a statement expressed by Predicate must have at least one object associated with Predicate. In our case, Ram is the required object with associated with predicate P.

PROPOSITIONAL CALCULUS

Given two numbers, we have various ways of combining them: add them, multiply them, etc. We can also take the negative or absolute value or square of a single number, and apply various functions to a given number. In other words, we can perform various operations on both individual numbers and on collections of numbers, and this endows the set of all numbers with a rich structure (e.g. arithmetic).

Propositions.

The first thing to do is to formally define what 'mathematical assertion' means. We shall refer to a mathematical assertion as a proposition; the book uses the word statement for this concept. Definition. A proposition is a statement that is either true or false, but not both, neither, or sometimes one and sometimes the other.

For example:

(1) Williams College is located in Williamstown. is a proposition (because it's true).

(2) Leo is a frog. is a propositions (because it's false).

(3) You are located in Williamstown. is not a proposition, because it's sometimes true and sometimes false.

(4) This statement is false. is not a proposition, because it is neither true nor false.

(5) Every even number larger than 2 is the sum of two primes. is a proposition, because it's either true or false.

Deductive Databases

A deductive database in SQL or any other database system is a tool that can draw conclusions about new facts based on the rules and information already present in the database. In deductive databases, datalog is the language commonly used to express facts, rules, and queries. The formula, when expressed in clausal form, consists of a number of clauses, each of which is made up of a number of literals joined exclusively by logical connectives marked with the OR symbol.

The following quantifiers are possible in a formula −

Universal quantifier − It may be read as "For all x, P(x) holds," which denotes that P(x) holds for all instances of x in the universe.

Trucks, for instance, all have wheels.

Existential quantifier − It means that P(x) holds for at least one item x in the universe and is expressed as "There exists an x such that P(x)".

UNIT - IV

XML Database

XML database is a data persistence software system used for storing the huge amount of information in XML format. It provides a secure place to store XML documents.

You can query your stored data by using XQuery, export and serialize into desired format. XML databases are usually associated with document-oriented databases.

Types of XML databases

There are two types of XML databases.

1. XML-enabled database
2. Native XML database (NXD)

XML-enable Database

XML-enable database works just like a relational database. It is like an extension provided for the conversion of XML documents. In this database, data is stored in table, in the form of rows and columns.

Native XML Database

Native XML database is used to store large amount of data. Instead of table format, Native XML database is based on container format. You can query data by XPath expressions.

Native XML database is preferred over XML-enable database because it is highly capable to store, maintain and query XML documents.

Let's take an example of XML database:

```
1. <?xml version="1.0"?>
2. <contact-info>
3.    <contact1>
4.       <name>Vimal Jaiswal</name>
5.       <company>SSSIT.org</company>
6.       <phone>(0120) 4256464</phone>
7.    </contact1>
8.    <contact2>
9.       <name>Mahesh Sharma </name>
10.            <company>SSSIT.org</company>
11.            <phone>09990449935</phone>
12.         </contact2>
13.      </contact-info>
```

In the above example, a table named contacts is created and holds the contacts (contact1 and contact2). Each one contains 3 entities name, company and phone.

XML Hierarchical (Tree) Data Model

The basic object is XML is the XML document.

There are two main structuring concepts that are used to construct an XML document:

Elements

Attributes

- Attributes in XML provide additional information that describe elements.

As in HTML, elements are identified in a document by their start tag and end tag.

The tag names are enclosed between angled brackets , and end tags are further identified by a backslash .

Complex elements are constructed from other elements hierarchically, whereas simple elements contain data values.

It is straightforward to see the correspondence between the XML textual representation and the tree structure.

In the tree representation, internal nodes represent complex elements, whereas leaf nodes represent simple elements,That is why the XML model is called a tree model or a hierarchical model.

Three main types of XML documents:

1. Data-centric XML documents : These documents have many small data items that follow a specific structure, and hence may be extracted from a structured database. They are formatted as XML documents in order to exchange them or display them over the Web.

2. Document-centric XML documents: These are documents with large amounts of text, such as news articles or books. There is little or no structured data elements in these documents.

3. Hybrid XML documents: These documents may have parts that contains structured data and other parts that are predominantly textual or unstructured.

Two types of XML

- Well-Formed XML
- Valid XML

Well-Formed XML

It must start with an XML declaration to indicate the version of XML being used—as well as any other relevant attributes.

Valid XML

- A stronger criterion is for an XML document to be valid.

- In this case, the document must be well-formed, and in addition the element names used in the start and end tag pairs must follow the structure specified in a separate XML DTD (Document Type Definition) file or XML schema file.

XML SCHEMA

An XML Schema describes the structure of an XML document, just like a DTD.

The XML Schema language is also referred to as XML Schema Definition (XSD).

XSD Example

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">


<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>


</xs:schema>
```

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

XML Querying

XQuery

XQuery uses XPath expressions, but has additional constructs. XQuery permits the specification of more general queries on one or more XML documents. The typical form of a query in XQuery is known as a FLWR expression, which stands for the four main clauses of XQuery and has the following form:

FOR <Variable bindings to individual nodes>

LET <Variable bindings to collections of nodes (elements)>

WHERE <qualifier conditions>

RETURN <query result specification>

XHTML was developed to make HTML more extensible and flexible to work with other data formats (such as XML). In addition, browsers ignore errors in HTML pages, and try to display the website even if it has some errors in the markup.

An XHTML document must have an XHTML <!DOCTYPE> declaration.

The <html>, <head>, <title>, and <body> elements must also be present, and the xmlns attribute in <html> must specify the xml namespace for the document.

Example

How to create a Hello World page in XHTML?

The Hello World page of XHTML looks like this

```
1. <?xml version="1.0" encoding="iso-8859-1"?>
2. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3. "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4. <html xml:lang="en" lang="en" xmlns="http://www.w3.org/1999/xhtml">
5. <head>
6. <title>Hello World</title>
7. </head>
8. <body>
9. <p>My first Web page.</p>
10.     </body>
11.     </html>
```

UNIT-V

Multimedia Database in DBMS

DBMS Multimedia Database

Multimedia database is a collection of multimedia data which includes text, images, graphics (drawings, sketches), animations, audio, video, among others. These databases have extensive amounts of data which can be multimedia and multisource. The framework which manages these multimedia databases and their different types so that the data can be stored, utilized, and delivered in more than one way is known as a multimedia database management system.

The multimedia database can be classified into three types. These types are:

1. Static media
2. Dynamic media
3. Dimensional media

The contents of a multimedia database management system can be:

1. **Media data:** It is the actual data which represents an object.
2. **Media format data:** The information such as resolution, sampling rate, encoding system, etc. about the format of the media data under consideration after is undergoes acquisition, processing, and encoding is the media format data.
3. **Media keyword data:** Media keyword data are the keyword description related to the generation of data. This data is also known as content descriptive data. Examples of content descriptive data are place, time, date of recording.

4. **Media feature data:** Media feature data contains data which is content dependent such as kind of texture, distribution of, and the different shapes present in the data.

The types of multimedia applications that are based on the data management characteristics are:

1. **Repository applications:** An extensive amount of multimedia data stored along with metadata for retrieval purposes.
2. **Presentation applications:** These involve the delivery of multimedia data subject to the temporal constraint. An optimal viewing or listening experience requires DBMS to deliver the data at a certain rate which offers the quality of service, which is above a particular threshold. This data is processed as it is being delivered.
3. **Collaborative work using multimedia information:** It involves the execution of a complex task by merging drawings and changing notifications.

This still leads to a number of challenges to multimedia databases. These are:

1. **Modelling:** Work in this area can improve the database versus information retrieval techniques.
2. Design: The physical, conceptual, and logical design of multimedia databases is not addressed entirely leading to performance and tuning issues.
3. **Storage:** The storage of databases on a standard disc can lead to problems like representation, mapping to disc hierarchies, compression, etc.
4. **Performance:** Audio-video synchronization and audio playback applications are where physical limitations dominate. Parallel processing can reduce these problems, but these techniques have not been completely developed yet. Multimedia databases also consume a lot of processing power and bandwidth.

5. **Queries and Retrieval:** Multimedia such as images, audio, video lead to retrieval and queries issues such as efficient query formation, query execution, etc.

Multimedia Database Applications:

1. **Documents and record management:** Industries which keep a lot of documentation and records. Ex: Insurance claim industry.
2. **Knowledge dissemination:** Multimedia database is an extremely efficient tool for knowledge dissemination and providing several resources. Ex: electronic books
3. **Education and training:** Multimedia sources can be used to create resources useful in education and training. These are popular sources of learning in recent days. Ex: Digital libraries.
4. **Real-time monitoring and control:** Multimedia presentation when coupled with active database technology can be an effective means for controlling and monitoring complex tasks. Ex: Manufacture control.

5. Marketing
6. Advertisement
7. Retailing
8. Entertainment
9. Travel

Multimedia database is database containing multimedia collections.

• Multimedia database management system is essential to manage multimedia data like text, graphics, animation, music, etc.

• Multimedia database management system can be defined as a software system that manages a collection of multimedia data and provides access to users to query and retrieve multimedia objects. • Generally,

multimedia database contains text, image, animation, video, audio, movie, sound etc. which is stored in binary form.

• SQL query language is used for query and retrieval of data.

• There are generally two types of multimedia databases

• Linked Multimedia Databases and Embedded Multimedia Databases.

• Linked multimedia databases In this database, multimedia elements are organized as image, audio/ MP3, video etc. All the data may be stored either on off-line sources (CD-ROM, Hard Disc, DVD etc.) or on Online sources. One great advantage of this type of database is that the size of database will be small due to the reason that multimedia elements are not embedded in the database, but only linked to it.

• Embedded multimedia database Embedded Multimedia Database implies that the database itself contains the multimedia objects as in the binary form in the database. The main advantage of such kind of database is that retrieval of data will be faster because of the reduced data access time. However, the size of the database will be very large.

Characteristics of MDBMS

• A MDBMS (Multimedia Database Management System) can be characterized based on its objectives at the time of handling multimedia objects.

• **Corresponding storage media**: Multimedia data must be stored and managed according to the specific characteristics of the available storage media.

• **Comprehensive search methods:** During a search in the database, an entry, given in the form of text or a graphical image, is found using different search queries and the corresponding search methods.

• **Format independent interface**: database queries should be independent of media format. MDBMS should provide information in formats requested by the application.

• **Simultaneous data access**: The same multimedia data can be accessed (even simultaneously) through different queries by several applications. Hence, consistent access to shared data can be implemented.

• **Management of large amount of data**: The MDBMS must be capable of handling and managing large amounts of data.

• **Long Transaction:** The performance of a transaction in a MDBMS means that transfer of a large amount of data will take a long time and must be done in a reliable manner.

• **Real-time Data:** The read and write operations of continuous data must be done in real-time. The data transfer of continuous data has a higher priority than other database management actions.

Operations of multimedia databases:

• Input (insert/record) operation: The data will be written to the database. The raw and registering data are always needed; descriptive data can be attached later.

• Output (play) operation: It involves reading the raw data from the database according to the registered data.

• Modification: It involves changing of raw, registering and descriptive data. Modification can also be understood as a data conversion from one format to another.

• Deletion Operation: This operation removes an entry from the database. The consistency of the data must be preserved.

Examples of multimedia database application areas:

- Digital Libraries.
- News-on-Demand.
- Video-on-Demand.
- Music database.
- Geographic Information Systems (GIS)
- Telemedicine.