UNIT I RELATIONAL DATABASES

Purpose of Database System – Views of data – Data Models – Database System Architecture – Introduction to relational databases – Relational Model – Keys – Relational Algebra – SQL fundamentals – Advanced SQL features – Embedded SQL– Dynamic SQL

UNIT-I / PART-A

1.	**Define database management system?** Database management system (DBMS) is a collection of interrelated data and a set of programs to access those data.
2.	**List any five applications of DBMS.** Banking, Airlines, Universities, Credit card transactions, Tele communication, Finance, Sales, Manufacturing, Human resources.
3.	**What is the purpose of Database Management System? *(Nov/Dec 14)*** Data redundancy and inconsistency , Difficulty in accessing data, Data isolation, Integrity problems, Atomicity problems and Concurrent access anomalies
4.	**Define instance and schema?** **Instance:** Collection of data stored in the data base at a particular moment is called an Instance of the database **Schema:** The overall design of the data base is called the data base schema.
5.	**Define the terms 1) physical schema 2) logical schema.** **Physical schema:** The physical schema describes the database design at the physical level, which is the lowest level of abstraction describing how the data are actually stored. **Logical schema:** The logical schema describes the database design at the logical level, which describes what data are stored in the database and what relationship exists among the data.
6.	**What is a data model? List the types of data models used?** A data model is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.
7.	**Define- relational algebra.** The relational algebra is a procedural query language. It consists of a set of operations that take one or two relation as input and produce a new relation as output.
8.	**What is a data dictionary?** A data dictionary is a data structure which stores meta data about the structure of the database i.e. the schema of the database.
9.	**List out the operations of the relational algebra** The Six basic operators Select, project, union, set difference, Cartesian product and Rename.
10.	**Define relational data model** Relational model use a collection of tables to represent both data and the relationships among those data. Each table has a multiple columns and each column has unique name.
11.	**Explain Semi structured data model** ● Specification of data where individual data item of same type may have different sets of attributes ● Sometimes called schema less or self-describing ● XML is widely used to represent this data model
12.	**Define Object based data model** Object based data model can be seen as extension of the E-R model with notion of encapsulation, methods and object identify.
13.	**Explain Hierarchical data model** ● The Hierarchical data model organizes data in a tree structure. There is hierarchy of parent and child data segments. ● This model uses parent child relationship. ● 1:M Mapping between record type

14.	**Define Network Model** ✓ Some data were more naturally modeled with more than one parent per child. ✓ This model permitted the modeling of M:N relationship
15.	**Write the characteristics that distinguish the Database approach with the File-based approach.** ***(Apr/May 15)(Nov/Dec 16)*** **File-based System.** 1. Separation and isolation of data 2. Duplication of data 3. Incompatible file formats 4. Data dependence 1. Control of data redundancy 2. Data consistency 3. Sharing of data 4. Improved data integrity 5. Improved security
16.	**What are the disadvantages of file processing system?*****(May/June 16)*** The file processing system has the following major disadvantages: ● Data redundancy and inconsistency Integrity Problems ● Security Problems Difficulty in accessing data Data isolation.
17.	**Define query language?** A query is a statement requesting the retrieval of information. The portion of DML that involves information retrieval is called a query language.
18.	**List the string operations supported by SQL?** **1)** Pattern matching Operation **2)** Concatenation **3)** Extracting character strings **4)** Converting between uppercase and lower case letters.
19.	**List out some date functions.** ✓ To_date ✓ To_char(sysdate,'fmt') ✓ d,dd,ddd,mon,dy,day,y,yy,yyy,yyyy,year,month,mm
20.	**What is the use of sub queries?** A sub query is a select-from-where expression that is nested with in another query. A common use of sub queries is to perform tests for set membership, make set comparisons, and determine set cardinality.
21.	**Name the categories of SQL command?** ***(May/June 16)*** SQL commands are divided in to the following categories: 1. Data - definition language 2. Data manipulation language 3. Data Query language 4. Data control language 5. Data administration statements 6. Transaction control statements
22.	**List the SQL domain Types?** SQL supports the following domain types. Char (n) , varchar (n), int , numeric (p,d) , float(n) , date.

23. **What are aggregate functions? And list the aggregate functions supported by SQL?**
Aggregate functions are functions that take a collection of values as input and return a single value. Aggregate functions supported by SQL are

- ✓ Average: avg
- ✓ Minimum: min
- ✓ Maximum: max
- ✓ Total: sum Count: count

24. **What is the difference between char and varchar2 data type?**

- ✓ Char and varchar2 are data types which are used to store character values.
- ✓ Char is static memory allocation; varchar2 is dynamic memory allocation.

25. **How to add primary key to a table with suitable query?**
Alter table <table name> add primary key(column);

26. **Differentiate static and dynamic SQL.** ***(Nov/Dec 14,15,16) (Apr/May 15)***

Static SQL	Static SQL
The SQL statements do not change each time the program is run is called Static SQL.	The SQL statements do not change each time the program is run is called Static SQL.
Static SQL is compiled and optimized prior to its execution	Static SQL is compiled and optimized prior to its execution
The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.	The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.

27. **Why does SQL allow duplicate tuples in a table or in a query result?** ***(Nov/Dec 15)***
If key constraint is not set on a relation every result in a relation will be considered as a tuple and hence SQL allows duplicate tuples in a table. Distinct keyword is used to avoid duplicate tuples in the result.

28. **What is embedded SQL? What are its advantages?**
The SQL standard defines embedded of SQL in a variety of programming languages such as C, Java, and Cobol. A language to which SQL queries are embedded is referred to as a host language, and the SQL structures permitted in the host language comprise embedded SQL. The basic form of these languages follows that of the System R embedding of SQL into
PL/I. EXEC SQL statement is used to identify embedded SQL request to the
preprocessor EXEC SQL <embedded SQL statement > END_EXEC

UNIT-I / PART-B

1.Explain the disadvantages in file systems.

The file processing system has a number of disadvantages:

- **Data redundancy and inconsistency**

Since different programmers create the files and application programs, the files will have a different format and the programs may be written in several programming languages. Also the same information may be duplicated in several files. For example the address and telephone number of a particular customer may appear in a file that consists of saving-account records and in a file that consist of checking account records. This **redundancy** leads to higher storage and access cost.

Data inconsistency occurs, if the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings account records but not elsewhere in the system.

- **Difficulty in accessing data**

Suppose that a bank officer needs to find the names of all customers who live in a particular postal-code area. The officer insists the data processing department to generate such a list. The original system has no application program to meet the request. However, the system has an application program to generate the list of all customers. The officer has only two choices: either can obtain the list of all customers and extract the needed information manually or ask the system programmer to write the necessary application program. Both are unsatisfactory.

Thus the conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner.

- **Data isolation**

  Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

- **Integrity problems**

  The data values stored in a database must satisfy some consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount (say, $50). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added it is difficult to change the programs to enforce them.

- **Atomicity Problems**

  A computer system like any other mechanical or electrical device is subject to failure. In many applications, if a failure occurs, it is difficult to restore the data to a consistent state as it existed prior to the failure. For example consider a program to transfer $50 from account A to account B. if a system failure occurs during the execution of the program, it is possible that $50 was removed from account A but was not credited to account B, resulting in inconsistent database state. The funds transfer must be atomic. It must happen in its entirety or not at all. Thus it is difficult to ensure atomicity in conventional file processing system.

- **Concurrent access anomalies**

  To improve the overall performance of the system, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data. For example consider bank account A, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from account at about the same time, the result of the concurrent executions may leave the account in an incorrect state. If the two programs run concurrently, they may both read the value $500, and write the result back $450 and $400 respectively. Depending on which one writes the value last, the account may contain either $450 or $400, rather than the correct value of $350.

- **Security Problems**

  Every user of the database system should not be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need to access information about customer accounts. The file processing systems do not enforce security constraints.

These are the difficulties that lead to the development of Database Management Systems.

These problems are solved by database management system.

2.What are different types of Data Models.Explain.

A Data Model is a collection of tools for describing

- Data
- Data relationships
- Data semantics and
- Data constraints

The **various Data Models** are

- ✓ Relational model
- ✓ Entity-Relationship data model (mainly for database design)

- ✓ Object-based data models (Object-oriented and Object-relational)
- ✓ Semi-structured data model (XML)

Relational Model

The relational model uses a collection of tables to represent both data and relationships among the data. Each table has multiple columns, and each column has a unique name. The below table called customer table, shows, for example, that the customer identified by customer-id 100 is named john and lives at 12 anna st. in Chennai and also shows his account number

Customer_id	Customer_name	Cutomer_street	Customer_city	Account_no
100	John	12 anna st	Chennai	A-101
101	Karthik	3 main st	Chennai	A201
103	Lilly	4 north st	Chennai	A-204

The relational model is an example of a **record-based model**. The record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type.

The relational model is at a lower level abstraction than the E-R model. Database designs are often carried out in the E-R model, and then translated to the relational model.

Entity-Relationship data model

The entity-relationship (E-R) data model, models an enterprise as a collection of *entities* and *relationships.*

Entity: is a "thing" or "object" in the enterprise that is distinguishable from other objects. They are described by a set of *attributes*

Relationship: is an association among several entities

The E-R model was developed to facilitate database design. The E-R model is very useful in mapping the meanings and interactions of real world enterprises onto a conceptual schema. The E-R model is represented diagrammatically by an *entity-relationship diagram* as shown below. In the **figure (b) customer** and **account** represents **entities**, the **ellipses** represent **attributes** and **depositor** represents **relationship** among the entities.

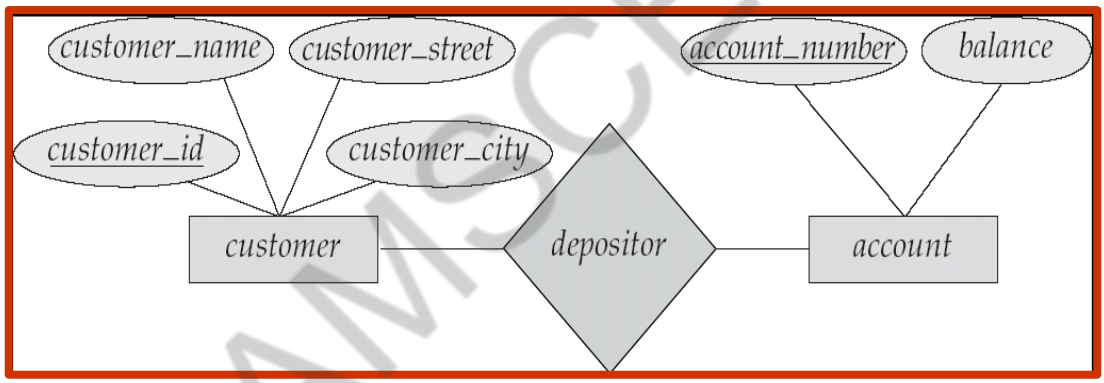


Figure b ER Model

Object based data models

Object based data models are categorized into object-oriented data model and object-relational data model. The **object–oriented data model** can be seen as **extending** the E-R model with notions of encapsulation, methods or functions and object identity. The **object-relational model** extends the relational data model by including object orientation and constructs to deal with added data types.

Semi-structured data model (XML)

Extensible markup Language is defined by the WWW Consortium (W3C). It was originally intended as a document markup language and not a database language. It has the ability to specify new tags, and to create nested tag structures which made XML a great way to exchange **data**, not just documents. XML has become the basis for all new generation data interchange formats. A wide variety of tools is available for parsing, browsing and querying XML documents/data

Hierarchical Model:

Hierarchical database model is one of the oldest models, dating from 1950's. The hierarchical model assumes that a tree structure is the most frequently occurring relationship. The hierarchical model organizes data elements as tabular rows, one for each instance of an entity. Consider a company's organizational structure. At the top we have general manager (GM). Under him he has a several deputy general managers (DGM). Each DGM take care of a couple of departments and each department will have a manger and many employees. When represented in hierarchical model there will be separate rows for representing the GM, each DGM, each Department, each Manager and each Employee. The row position implies a relationship to other rows. A given employee belongs to the department that is the closest above it in the list and the department belongs to the manager immediately above it and so on.

It shows the hierarchical model of data for a sales order processing application.

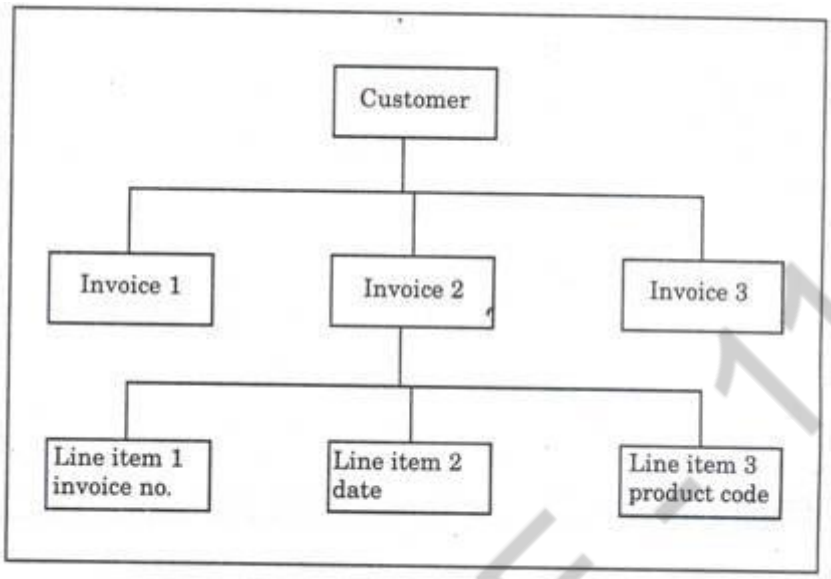
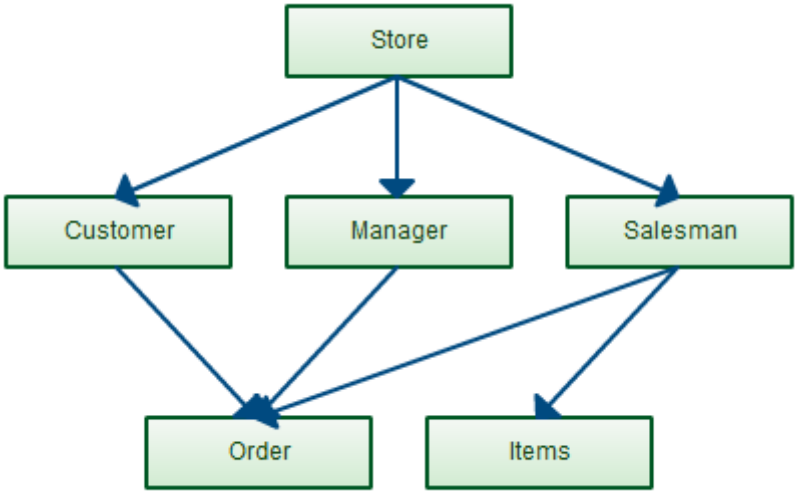


Fig. 9.4 Hierarchical model of data for sales order processing

Network Model:

The network model replaces the hierarchical tree with a graph, thus, allowing more general connections among the nodes. The main difference of the network model from the hierarchical model is its ability to handle many-to-many relationships. In other words, it allows a record to have more than one parent. Suppose an employee works for two departments. The strict hierarchical arrangement is not possible here and the tree becomes a more generalized graph-a network. Logical proximity fails because it is not possible to place a data item simultaneously in two locations in the list. Although it is possible to handle such situations in a hierarchical model, it becomes more complicated and difficult to comprehend. The network model was evolved to specifically handle non-hierarchical relationships.

In network database terminology, a relationship is a set. Each set is made of at least two types of records: an owner record and a member record.



3.Explain the functions of database administrator.

Database Administrator: The person who has central control over the system is called database administrator (DBA). The functions of DBA include:

- **Schema Definition:** the DBA creates the original database schema by executing a set of data definition statements in the DDL.
- **Storage Structure and access-method definition:** the DBA is also responsible for defining the storage structure and access methods.
- **Schema and physical organization modification:** the DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
- **Granting of authorization for data access:** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.
- **Routine Maintenance:** The DBA,

7. Periodically makes the backup of the database, stores either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
8. Ensures that enough disk space is available for normal operations, and upgrades disk space as required.
9. Monitors jobs running on the database and ensures that the performance is not degraded by very expensive tasks submitted by some users.

4. Explain DATABASE SYSTEM STRUCTURE(OR)COMPONENTS OF DBMS or dbms architecture.(May/June 2016,2017,Nov/Dec 2017)

The functional components of a database system can be broadly divided into two Categories namely,

- Storage manager components
- Query Processor components

Database Users and interfaces

There are four different types of database-system users. Different types of interfaces have been designed for the different types of users.

- **Naïve Users** are unsophisticated users who interact with the system by invoking one of the application

programs that have been written previously. The typical user interface for naïve users is a forms interface where the user can fill in appropriate fields of the form. Naïve users may also simply read reports generated from the database.

- **Application Programmers** are computer professionals who write application programs. Application Programmers can choose from many tools to develop user interfaces. Rapid Application Development (RAD) tools are tools that enable an application programmer to construct forms and reports without witting a program.

- **Sophisticated Users** interact with the system without writing programs. Instead they form requests in a database query language. They submit such queries to query processor, whose function is to break down DML statements into instructions that the storage manager understands.

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

- **Database Administrator:** The person who has central control over the system is called database administrator (DBA).

Functional Components

Storage Manager

Storage manager is important because databases typically require a large amount of storage space. 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. The storage manager translates the various DML statements into low-level file system commands. Thus, the storage manager is responsible for storing retrieving, and updating data in the database.

The **storage manager components** include:

5) **Authorization and integrity manager**

It tests for the satisfaction of integrity constraints and checks the authority of users to access data.

6) **Transaction Manager**

It ensures that the database remains in a consistent state despite system failures, and that concurrent transaction executions proceed without conflicting.

7) **File Manager**

It manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

8) **Buffer Manager**

It is responsible for fetching data from disk storage into main memory, and decides what data to cache in main memory.

The **storage manger** implements several **data structures** as part of the physical system, implementation:

6. **Data files**, which store the database itself.
7. **Data Dictionary**, which stores structure of the database called metadata.
8. **Indices**, which provide fast access to data items that hold particular values.

Query Processor

The **query processor components** include

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

- **DML compiler** translates DML statements in a query language into an evaluation

plan consisting of low-level instructions that the query evaluation engine understands. The DML compiler also performs query optimization that is it picks the lowest cost evaluation plan from among the alternatives.

- **Query evaluation engine** executes low-level instructions generated by the DML compiler.

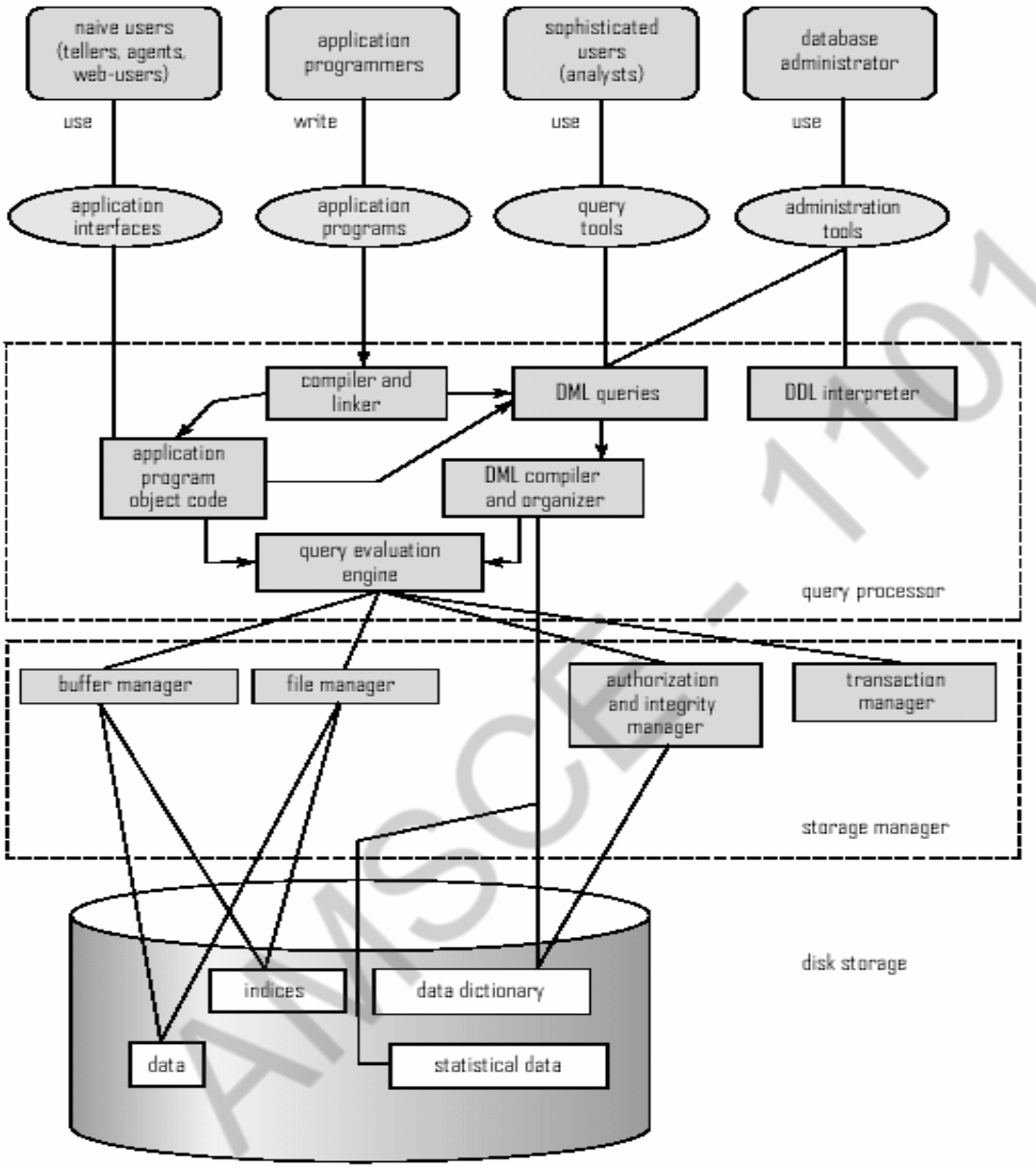


Figure 1. Database System Structure

5. Explain various Relational Algebra operations with examples.(Apr/May 2017,May/June 2016)

The relational Algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result.

Fundamental Operations:

The fundamental Operations in Relational Algebra are:

- select
- project
- union
- set difference
- cartesian product
- rename

Here the select, project and rename operations are called unary operations, because they operate on one relation.

The other three operations operate on pairs of relations and are, therefore called binary operations.

The select operation:

The select operation selects tuples that satisfy a given predicate. The lower case Greek letter sigma (σ) is used to denote the selection. The predicate appears as the subscript to σ.

The Project Operation:

The project operation is a unary operation that returns its argument relation, with certain attributes left out. Projection is denoted by the Greek letter pi (Õ). The attributes that should appear in the result are listed as subscript to the Õ. The argument relation follows in the parenthesis.

Example:
loan relation:

loan_number	*branch_name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

Question:

List all the loan-numbers and amount of the loan

Relational algebra query:

The equivalent relational algebra query for the given question is,

$Õ_{\text{loan-number, amount}}$ (loan)

Result:

The result of the query is given below.

loan_number	*amount*
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

Composition of relational operations:

The relational operations can be composed together into a relational algebra expression.

Example:

customer relation:

customer_name	customer_street	customer_city
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

Question:

Find those customers who live in Harrison.

Relational algebra query:

The equivalent relational algebra query for the given question is,

$\tilde{O}_{\text{customer-name}}(\sigma_{\text{customer-city} = \text{“Harrison”}}(\text{customer}))$

Here the expression is given as argument to the projection operation instead of the name of a relation.

Result:

Customer-name
hayes
jones

The Union Operation:

The union operation is the binary operation which combines two relations. The union operation is denoted by the letter (È). For a union operation r È s to be valid, two conditions must hold.

The relations r and s must be of the same arity. That is they must have the same number of attributes.

The domains of the ith attribute of r and ith attribute of s must be the same for all i.

Example:

Consider the borrower and depositor relation

borrower relation:

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

depositor relation:

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

Question:

Find the names of all customers who have either an account or a loan or both?

Relational algebra query:

The equivalent relational algebra query for the given question is,

$\tilde{O}_{\text{customer-name}}$ (borrower) È $\tilde{O}_{\text{customer-name}}$ (depositor)

Result:

customer_name
Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

The set difference operation:

The set-difference operation denoted by (–), finds the tuples that are in one relation but are not in another. It is a binary operation. The expression r–s produces a relation containing those tuples in r but not in s. For a set difference operation r–s to be valid , the relations r and s should be of the same arity and the domains of the ith attribute of r and ith attribute of s must be the same.

Example:

Consider the depositor and borrower relation. Refer union operation.

Question:

Find all the customers of the bank who have an account but not a loan?

Relational algebra query:

The equivalent relational algebra query for the given question is,

$\tilde{O}_{\text{customer-name}}$ (depositor) – $\tilde{O}_{\text{customer-name}}$ (borrower)

Result:

customer_name
Johnson
Lindsay
Turner

The Cartesian product operation:

The Cartesian product operation, denoted by (**X),** allows combining information from any two relations.

The cartesian product of r1 and r2 is written as r1X r2. Since the same attribute name may appear in both r1 and r2, a naming schema must be used to distinguish between these attributes. This is done by attaching the name of the relation to the attribute from which the attribute originally came. The naming convention requires that the relations that are the arguments of the Cartesian product must have distinct names.

If r1 contains n1 tuples and r2 contains n2 tuples, then there are n1*n2 ways of choosing a pair of tuples-one tuple from each relation. So, there are n1*n2 tuples in r.

Example:

Consider the borrower and loan relations given below.

borrower relation:

customer-name	loan-no
Adams	L-16
Hayes	L-93
Jackson	L-15

Loan relation:

loan-no	branch-name	amt
L-93	Roundhill	900
L-16	Downtown	1500
L-15	Perryridge	1300

Question:

Find the names of all customers who have a loan at the perryridge branch.

Computation of Relational algebra query:

This question needs the information in both the loan relation and the borrower relation.

If the query is written as

$\sigma_{\text{branch-name} = \text{“perryridge”}}$ (borrower X loan)

Result of borrower X loan:

customer name	borrower. loan –number	loan. loan-number	branch-name	amount
Adams	L-16	L-93	Round Hill	900
Adams	L-16	L-16	Downtown	1500
Adams	L-16	L-15	Perryridge	1300
Hayes	L-93	L-93	Round Hill	900
Hayes	L-93	L-16	Downtown	1500
Hayes	L-93	L-15	Perryridge	1300
Jackson	L-15	L-93	Round Hill	900
Jackson	L-15	L-16	Downtown	1500
Jackson	L-15	L-15	Perryridge	1300

Result of $\sigma_{\text{branch-name} = \text{“perryridge”}}$ (borrower X loan):

customer-name	borrower. loan –number	loan. loan-number	branch-name	amount
Adams	L-16	L-15	Perryridge	1300
Hayes	L-93	L-15	Perryridge	1300
Jackson	L-15	L-15	Perryridge	1300

The **above relation** pertains results to only perryridge branch. However, the customer-name column may

contain customers who do not have a loan at the perryridge branch. Therefore to obtain the correct result the query has to be written as below.

$\sigma_{\text{borrower.loan-number = loan.loan-number}}$
($\sigma_{\text{branch-name = "perryridge"}}$ (borrower X loan))

The **result** of the above query is given below:

customer - name	borrower. loan –number	loan. loan-number	branch-name	amount
Jackson	L-15	L-15	Perryridge	1300

Finally since only the customer-name is needed the projection operation is used as below.
$\Pi_{\text{customer-name}}(\sigma_{\text{borrower.loan-number = loan.loan-number}}$
($\sigma_{\text{branch-name = "perryridge"}}$ (borrower X loan)))

Final result:

Customer-name
Jackson

The Rename Operation:

The rename operator is used to rename the attributes. The rename operator is denoted by the lowercase Greek letter rho (r). Given a relational algebra expression E, the expression

$r_x(E)$ returns the result of expression E under name x.

A second form rename operation is as follows.

$r_x(A1,A2,\ldots,An)(E)$

Returns the result of expression E under the name x, and with the attributes renamed to A1, A2,….,An.

Examples:

Consider the **account relation**.

acc-no	branch-name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900

6.Explain in detail about Data Definition Language (DDL) Commands.(May/June 2016,Nov/Dec 2017)

The SQL data definition language allows specification of not only a set of relations but also information about each relation, including

- ✓ The schema for each relation.
- ✓ The domain of values associated with each attribute.
- ✓ Integrity constraints
- ✓ The set of indices to be maintained for each relation.
- ✓ Security and authorization information for each relation.

The physical storage structure of each relation on disk

Creating Tables:

The SQL command for creating an empty table has the following form:

Syntax:

create table <table-name> (<column 1> <data type> ,<column2t> <data type>, . . . , .<column n> <data type>, <integrity constraint1>,…., <integrity constraint n>);

For each column, a name and a data type must be specified and the column name must be unique within the table definition. Column definitions are separated by comma. There is no difference between names in lower case letters and names in upper case letters. In fact, the only place where upper and lower case letters matter are strings comparisons.

Example:

The create table statement for EMP table has the form

create table EMP (EMPNO number(4), ENAME varchar2(30), JOB varchar2(10), MGR number(4), HIREDATE date, SAL number(7,2), DEPTNO number(2));

Deleting tables:

The following command will drop the table from the database.

Syntax:

drop table <table-name>;

Example:

drop table emp;

Alter table:

Alter table command is used to add, modify or drop attributes or columns from the table.

Syntax:

alter table <table-name> add/modify <col-name1> <datat ype>,….,<col-namen> <data type>;

alter table <table-name> drop column <col-name>;

Example:

alter table emp add year char(3);

Truncate table:

The truncate table command deletes the rows in the table but retains the table structure.

Syntax:

truncate table <table-name>;

Example:

Truncate table emp;

7.Explain in detail about Data Manipulation Language(DML) commands.(May/June 2016,Nov/Dec 2017)

Insert command:

Insert command is used to insert single or multiple rows in the table.

Syntax:

Single row:

insert into <table-name> values (list of values);

Multiple rows:

insert into <table-name> values(&val1,'&val2',….,&valn);

Example:

Single row:

insert into emp values(1000,'anitha','lecturer',400,'12-aug-80',10000,100);

Multiple rows:

insert into emp values(&empno,'&empname','&job',&mgr,'&hiredate',&salary,&deptno);

Delete command:

The delete command removes tuples from a relation. It includes the where clause to select the tuples to be deleted.

Syntax:

delete from <table-name> where attribute-name = 'value' or value;

Example:

delete from emp where empname = 'anitha';

Update command:

The update command is used to modify attribute values of one or more selected tuples. It includes a where clause to select the tuples to be modified from a single relation.

Syntax:

update <table-name>

set col1=val1,…,coln = valn

where condition;

Example:

update emp

set salary = 12000

where empno =1000;

Select clause:

The select clause is used to select a particular or all attributes from relations.

Syntax:

Select <attribute list> or (*) from <table list> where <condition>;

Example:

Select * from emp where empno =1000;

Select empno, empname from emp;

8.Explain in detail about Transaction control language(TCL) commands.(May/June 2016,Nov/Dec 2017)

Transaction management ensures the atomicity and consistency of all transactions. A transaction must be successfully completed after it has started, or SQL server undoes all the data modifications made since the start of the transaction. Some transaction statements are,

Commit:

A commit ends the current transaction and makes permanent any changes made during the transaction.

Syntax:

commit;

Rollback:

A rollback does exactly the opposite to commit. It ends the transaction but undoes any changes made during the transaction.

Syntax:

rollback to **savepoint** <savepointname>;

Savepoint:

Savepoint marks and saves the current point in the processing of a transaction. When a savepoint is used with a rollback statement, parts of a transaction can be undone.

savepoint <savepointname>;
savepoint s1;

9.Explain in detail about Data control language(DCL) commands.(May/June 2016,Nov/Dec 2017)

The DCL language is used for controlling the access to the table and hence securing the database. DCL is used to provide certain privileges to a particular user. Privileges are rights to be allocated. The privilege commands are namely,
1)Grant
2)Revoke

GRANT COMMAND:It is used to create users and grant access to the database. It requires database administrator (DBA) privilege, except that a user can change their password. A user can grant access to their database objects to other users.

REVOKE COMMAND:Using this command , the DBA can revoke the granted database privileges from the user.

SYNTAX

GRANT COMMAND

Grant < database_priv [database_priv…..] > to <user_name> identified by <password> [,<password…..];
Grant <object_priv> | All on <object> to <user | public> [With Grant Option];

REVOKE COMMAND

Revoke <database_priv> from <user [, user] >;
Revoke<object_priv> on <object> from < user | public >;

<database_priv> -- Specifies the system level priveleges to be granted to the users or roles. This includes create / alter / delete any object of the system.
<object_priv> -- Specifies the actions such as alter / delete / insert / references / execute / select / update for tables.
<all> -- Indicates all the priveleges.
[With Grant Option] – Allows the recipient user to give further grants on the objects.
The priveleges can be granted to different users by specifying their names or to all users by using the “Public” option.

GRANT

(i) Grant all on employees to abcde;

(ii) Grant select , update , insert on departments to abcde with grant option;

(iii)grant alter,update,insert on empl to deepa;

REVOKE
(i) Revoke all on employees from abcde;

(ii) Revoke select , update , insert on departments from abcde;

(iii) revoke alter,update,insert on empl from deepa;

10.Explain various types of integrity constraints.(Nov/Dec 2017)

Data Integrity
Data Integrity validates the data before getting stored in the columns of the table.
SQL Server supports four type of data integrity:

a)Domain Integrity constraints

Domain integrity validates data for a column of the table.

Example: Create table student(regno number(10),name varchar(25));

This constraint denotes datatype of column(field or attribute) name and maximum size of the columnname denoted with ( ).ie (10) and (25)

Datatype may be of different types such as number,varchar,char,integer,float,numeric,date.

b)Entity integrity constraints

1.Null
2.Not Null
3.Primary key
4.Unique key

Null: allow null values
e.g Create table employee(empid number(10),firstname varchar(20),middlename varchar(20) null,lastname varchar(20));
Not Null: will not allow null values

e.g Create table bankaccount(accno number(10) not null,customername varchar(20));

Primary Key:avoids duplicate values during insertion of values.
It won't allow null values.

e.g Create table bankaccount(accno number(10) Primary key,customername varchar(20));

Unique Key: avoids duplicate values during insertion of values.
It allows null values.

e.g.Create table student(regno number(10),name varchar(25),pancardno number(10) unique);

c) **Referential integrity Constraints:**

An RDBS needs to maintain and have logical relationships between various tables for it to work efficiently. When such relationships exist and are maintained as they should be, it represents referential integrity of the data.

e.g.

First table:

Create table employeetable1(empid number(8) primary key,empname varchar(20));

Second table:

Create table employeetable2(empno number(8) primary key,phoneno number(10),**foreign key(empno) references employeetable1(empid));**

Foreign key ensures that empno values in second table are same with the already inserted empid values in first table.Otherwise the empno values cannot be inserted.

11.Explain in detail about view.(8 marksor 6 marks)(May/June 2016)

View is nothing but false table.The needed data from more than one table can be taken and created as view. A view on the other hand does not exist on its own right. It is a named table that is represented not by its own physically separate stored data, but by its definition in terms of other named tables.

create view v **as** <query expression>

where <query expression> is any legal query expression. The view name is represented by v.

EXAMPLE

1)Create view s1 as select studentname from student where CGPA>7.5;

2)Create view s2 as select regno,studentname,cgpa from student;

12.How will you secure your database (or)Explain about triggers.

A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database. Triggers are for securing the database.The parts of a trigger are,

Trigger statement: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.

Trigger body or trigger action: It is a PL/SQL block that is executed when the triggering statement is used.

Trigger restriction: Restrictions on the trigger can be achieved

TYPES OF TRIGGERS

The various types of triggers are as follows,

Before: It fires the trigger before executing the trigger statement.

After: It fires the trigger after executing the trigger statement.

For each row: It specifies that the trigger fires once per row.

For each statement: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

VARIABLES USED IN TRIGGERS

:new

:old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation

SYNTAX

create or replace trigger triggername [before/after] {insert or update or delete}

on [tablename] [for each row/statement]

begin

DML OPERATION QUERY STATEMENT
end;

EXAMPLE

CREATING TRIGGER FOR UPDATE OPERATION

```
create or replace trigger triggerupdating after update on employee for each row
begin
update employee set salary = 12000 where empno =1000;
end;
/
```

Any user cannot able to update the salary on employee table after the triggerupdating is created.

13.Explain in detail about Embedded SQL.(May/June 2016,Apr/May 2017)

*)The SQL standard defines embeddings of SQL in a variety of programming languages such as Pascal, PL/I, FORTRAN, C, and COBOL.

*)A language to which SQL queries are embedded is referred to as a ***host language***, and the SQL structures permitted in the host language comprise ***embedded SQL.***

*)Programs written in the host language can use the embedded SQL syntax to access and update data stored in a database.

(i)The **open** statement causes the query to be evaluated and to save the result in the temporary relation

```
Open q_cursor;
```

(ii)The **fetch** statement causes the values of one tuple in the query result to be placed on host language variables.

```
fetch q_cursor into q_val;
```

(iii)The **close** statement causes the database system to delete the temporary relation that holds the result of the query.

```
Close q_cursor;
```

example:

```
declare
 pi number:=3.14;
area number;
 cursor q_cursor is
 select * from q;
 q_val q_cursor%rowtype;
 begin
 open q_cursor;
 fetch q_cursor into q_val;
 area:=pi*power(q_val.radius,2);
insert into q1 values(q_val.radius,area);
close q_cursor;
end;
/
```

14.Explain different types of SQL Joins with examples.

The relationship between the two tables is specified by the customer_id key, which is the "primary key" in customers table and a "foreign key" in the orders table:

customer_id	first_name	last_name	email	address	city	state	zipcode
1	George	Washington	gwashington@usa.gov	3200 Mt Vernon Hwy	Mount Vernon	VA	22121
2	John	Adams	jadams@usa.gov	1250 Hancock St	Quincy	MA	02169
3	Thomas	Jefferson	tjefferson@usa.gov	931 Thomas Jefferson Pkwy	Charlottesville	VA	22902
4	James	Madison	jmadison@usa.gov	11350 Constitution Hwy	Orange	VA	22960
5	James	Monroe	jmonroe@usa.gov	2050 James Monroe Parkway	Charlottesville	VA	22902

order_id	order_date	amount	customer_id
1	07/04/1776	$234.56	1
2	03/14/1760	$78.50	3
3	05/23/1784	$124.00	2
4	09/03/1790	$65.50	3
5	07/21/1795	$25.50	10
6	11/27/1787	$14.40	9

Note that (1) not every customer in our customers table has placed an order and (2) there are a few orders for which no customer record exists in our customers table.

Inner Join

Let's say we wanted to get a list of those customers who placed an order and the details of the order they placed. This would be a perfect fit for an inner join, since an inner join returns records at the intersection of the two tables.

select first_name, last_name, order_date, order_amount
from customers c
inner join orders o
on c.customer_id = o.customer_id
first_name
last_name
order_date
order_amount

George
Washington
07/4/1776
$234.56
John
Adams
05/23/1784
$124.00
Thomas
Jefferson
03/14/1760
$78.50
Thomas
Jefferson
09/03/1790
$65.50

Note that only George Washington, John Adams and Thomas Jefferson placed orders, with Thomas Jefferson placing two separate orders on 3/14/1760 and 9/03/1790.

Left outer Join

If we wanted to simply append information about orders to our customers table, regardless of whether a customer placed an order or not, we would use a left join. A left join returns all records from table A and any matching records from table B.

```
select first_name, last_name, order_date, order_amount
from customers c
left join orders o
on c.customer_id = o.customer_id
```

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	$234.56
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	03/14/1760	$78.50
Thomas	Jefferson	09/03/1790	$65.50
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

Note that since there were no matching records for James Madison and James Monroe in our orders table, the order_date and order_amount are NULL, which simply means there is no data for these fields.

So why would this be useful? By simply adding a “where order_date is NULL” line to our SQL query, it returns a list of all customers who have not placed an order:

```
select first_name, last_name, order_date, order_amount
from customers c
left join orders o
```

on c.customer_id = o.customer_id

where order_date is NULL

Right outer Join

Right join is a mirror version of the left join and allows to get a list of all orders, appended with customer information.

select first_name, last_name, order_date, order_amount

from customers c

right join orders o

on c.customer_id = o.customer_id

first_name
last_name
order_date
order_amount
George
Washington
07/04/1776
$234.56
Thomas
Jefferson
03/14/1760
$78.50
John
Adams
05/23/1784
$124.00
Thomas
Jefferson
09/03/1790
$65.50
NULL
NULL
07/21/1795
$25.50
NULL
NULL
11/27/1787
$14.40

Note that since there were no matching customer records for orders placed in 1795 and 1787, the first_name and last_name fields are NULL in the resulting set.

Also note that the order in which the tables are joined is important. We are right joining the orders table to the customers table. If we were to right join the customers table to the orders table, the result would be the same as left joining the orders table to the customers table.

Why is this useful? Simply adding a “where first_name is NULL” line to our SQL query returns a list of all orders for which we failed to record information about the customers who placed them:

select first_name, last_name, order_date, order_amount

from customers c

right join orders o

on c.customer_id = o.customer_id

where first_name is NULL

Full outer Join :Finally, for a list of all records from both tables, we can use a full join.

select first_name, last_name, order_date, order_amount

from customers c

full join orders o

on c.customer_id = o.customer_id

first_name
last_name
order_date
order_amount
George
Washington
07/04/1776
$234.56
Thomas
Jefferson
03/14/1760
$78.50
John
Adams
05/23/1784
$124.00
Thomas
Jefferson
09/03/1790
$65.50
NULL
NULL
07/21/1795
$25.50
NULL
NULL
11/27/1787
$14.40
James
Madison
NULL
NULL
James
Monroe
NULL
NULL

UNIT-II / PART-A	
1.	**Explain entity relationship model?*(May/June 16)*** The entity relationship model is a collection of basic objects called entities and relationship among those objects. An entity is a thing or object in the real world that is distinguishable from other objects.
2.	**What is relationship? Give examples** A relationship is an association among several entities. Example: A depositor relationship associates a customer with each account that he/she has.
3.	**What are stored and derived attributes?** Stored attributes: The attributes stored in a data base are called stored attributes. Derived attributes: The attributes that are derived from the stored attributes are called derived attributes
4.	**What are composite attributes?** Composite attributes can be divided in to sub parts. The degree of relationship type is the number of participating entity types.
5.	**What is a weak entity? Give example. *(Nov/Dec 16)*** It is an entity that cannot be identified uniquely without considering some primary key attributes of another identifying owner entity. An example is including Dependent information for employees for insurance purposes.
6.	**What are attributes? Give examples.** An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set. Example: possible attributes of customer entity are customer name, customer id, Customer Street, customer city.
7.	**Mention the 2 forms of integrity constraints in ER model?** ✓ Key declarations ✓ Form of a relationship

8.	**What is the use of integrity constraints?** Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency. Thus integrity constraints guard against accidental damage to the database
9.	**List some security violations (or) name any forms of malicious access.** 1) Unauthorized reading of data 2) Unauthorized modification of data 3) Unauthorized destruction of data.
10.	**What is a primary key?** Primary key is a set of one or more attributes that can uniquely identify record from the relation; it will not accept null values and redundant values. A relation can have only one primary key.
11.	**What is called query processing?** Query processing refers to the range of activities involved in extracting data from a database.
12.	**What is called a query evaluation plan?** A sequence of primitive operations that can be used to evaluate be query is a query evaluation plan or a query execution plan.
13.	**What is called as an N-way merge?** The merge operation is a generalization of the two-way merge used by the standard in-memory sort-merge algorithm. It merges N runs, so it is called an N-way merge.
14.	**What is a super key?** A super key is a set of one or more attributes that collectively allows us to identify uniquely an entity in the entity set.

15.	**What is foreign key?** A relation schema r1 derived from an ER schema may include among its attributes the primary key of another relation schema r2.this attribute is called a foreign key from r1 referencing r2.
16.	**What is the difference between unique and primary key?** Unique and primary key are keys which are used to uniquely identify record from the relation. But unique key accepts null values; primary key does not accept null values.
17.	**What does the cardinality ratio specify?** Mapping cardinalities or cardinality ratios express the number of entities to which another entity can be associated. Mapping cardinalities must be one of the following: One to one, One to many, Many to one and Many to many.
18.	**Explain the two types of participation constraint.** Total: The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R. ✓ Partial: if only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be partial.
19.	**Define Tuple variable?** Tuple variables are used for comparing two tuples in the same relation. The tuple variables are defined in the from clause by way of the as clause.
20.	**What is first normal form?** The domain of attribute must include only atomic (simple, indivisible) values.
21.	**What is 2NF?** Relation schema R is in 2NF if it is in 1NF and every non-prime attribute An in R is fully functionally dependent on primary key.
22.	**What is meant by domain key normal form?** Domain/key normal form (DKNF) is a normal form used in database normalization which requires that the database contains no constraints other than domain constraints and key constraints.

23.	**Define Functional dependency.** ***(Apr/May 15)*** In a given relation R, X and Y are attributes. **Attribute Y is functionally dependent on attribute X if each value of X determines EXACTLY ONE value of Y**, which is represented as X → Y (X can be composite in nature). We say here "x determines y" *or "y is functionally dependent on x" Empid → Ename*
24.	**Define full functional dependency.** The removal of any attribute A from X means that the dependency does not hold any more.
1.	**Explain Entity Relationship model with examples.(Apr/May 2017)** The entity-relationship (E-R) data model, models an enterprise as a collection of *entities* and *relationships.* **Entity:** is a "thing" or "object" in the enterprise that is distinguishable from other objects. They are described by a set of *attributes* **Relationship:** is an association among several entities **1.1 Attributes and Types of Attributes:** **Attributes:** An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set. For example possible attributes of the customer entity set are customer-id, customer-name, customer-street, and customer-city. Each entity has a value for each attribute. For instance, a particular customer entity may have the value 100-00-20 for customer-id, the value john for customer-name etc. **Types of attributes:** **Simple attribute:** The attributes that cannot be divided into sub parts is called as simple attribute.

Example:

A person's age cannot be divided into sub parts.

Composite attributes:

Composite attributes are attributes that can be divided into sub parts. Composite attributes can also be used as a hierarchy. The divided sub parts are called as component attributes.

Example:

An attribute name could be structured as a composite attribute consisting of first-name, middle-initial, last-name.

Single-valued attributes:

The attribute that has only one value for a particular entity is called as single-valued attributes.

Example:

A person can have only one date of birth.

Multivalued attributes:

The attribute that has zero or one or more values is called as multivalued attribute.

Example:

A person can have zero or one or more phone numbers.

Derived attributes:

The attribute that derives its value from another attribute (stored or base attribute)is called as derived attribute.

Example:

A person's age can be derived from the person's date of birth.

Age-Derived attribute

DOB-Stored attribute

1.2 Relationship Sets:

A **relationship** is an association among several entities. A **relationship set** is the set of relationships of the same type. A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

$$\{(e1, e2, \ldots en) \mid e1 \in E1, e2 \in E2, \ldots, en \in En\}$$

where ($e1$, $e2$, …, en) is a relationship

Example:

Consider two entity sets customer and loan in figure below. The association between these two entity sets can be expressed using the relationship set borrower. The **figure (c)** depicts this association.

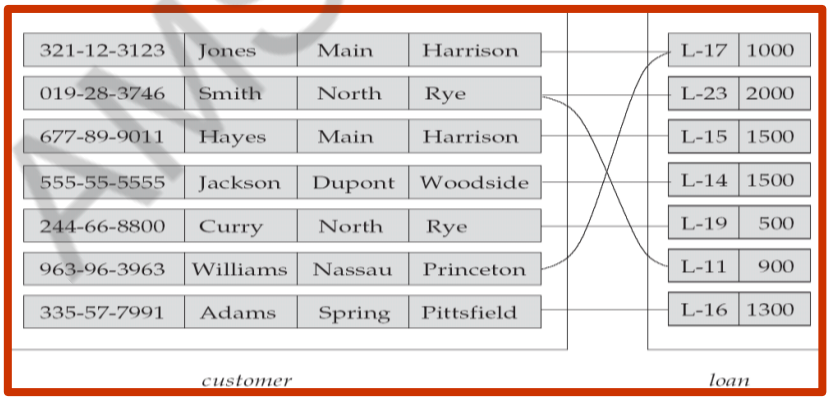


Figure c: relationship set borrower

The association between entity sets is referred to as **participation**. The function that an entity plays in a relationship is called that **entity's role.**

A relationship may also have attributes called **descriptive attributes**. For example consider a relationship set depositor with entity sets customer and account. It can be associated with the attribute access-date to specify the most recent date on which a customer accessed an account. **Figure(d)** depicts the example.

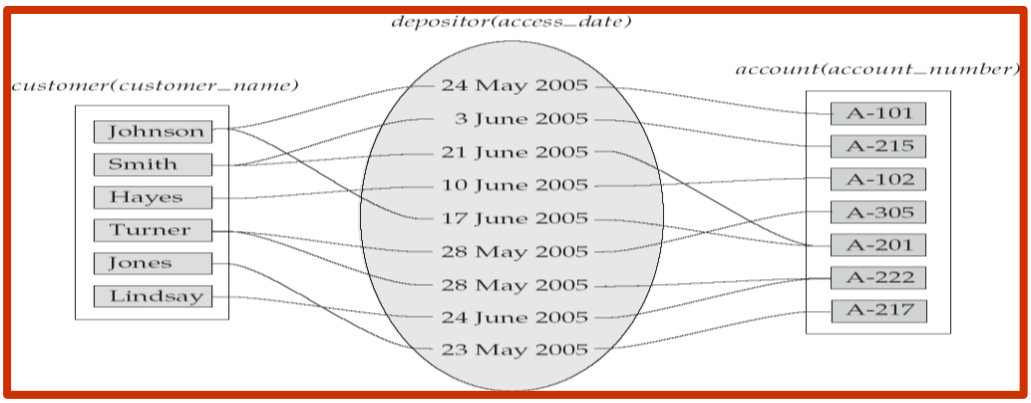


Figure d: Access-date as descriptive attribute of the depositor relationship set

Most of the relationship sets in a database system are binary. Occasionally, however, relationship sets involve more than two entity sets. The number of entity sets that participate in a relationship set is called the degree of the relationship set. A binary relationship set is of degree 2; a ternary relationship set is of degree 3.

1.3 Constraints:

An E_R enterprise schema may define certain constraints to which the contents of a database must conform. Two important constraints are: **mapping cardinalities** and **participation constraints.**

Mapping Cardinalities:

Mapping cardinalities or cardinality ratios express the number of entities to which another entity can be associated via a relationship set. They are most useful in describing binary relationship sets.For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following:

One to one: An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A. The **next figure**, shows one to one relation.

One to many: An entity in A is associated with any number of entities in B. An entity in B, however, can be associated with at most one entity in A. The **below figure**, shows one to many relation.

B

Many to one: An entity in A is associated with at most one entity in B. An entity in B, however can be associated with any number of entities in A. the **below figure** shows Many to one relationship

Many to many: An entity in A is associated with any number of entities in B, and an entity in B is associated with any number of entities in A. the **below figure** shows many to many relation.

Participation Constraints:

Total Participation:

The participation of an entity set E in a relationship set R is said to be **total** if every entity in E participates in at least one relationship in R.

Example: every loan entity should be related to at least one customer through the borrower relationship. Therefore the participation of loan in the relationship set borrower is total.

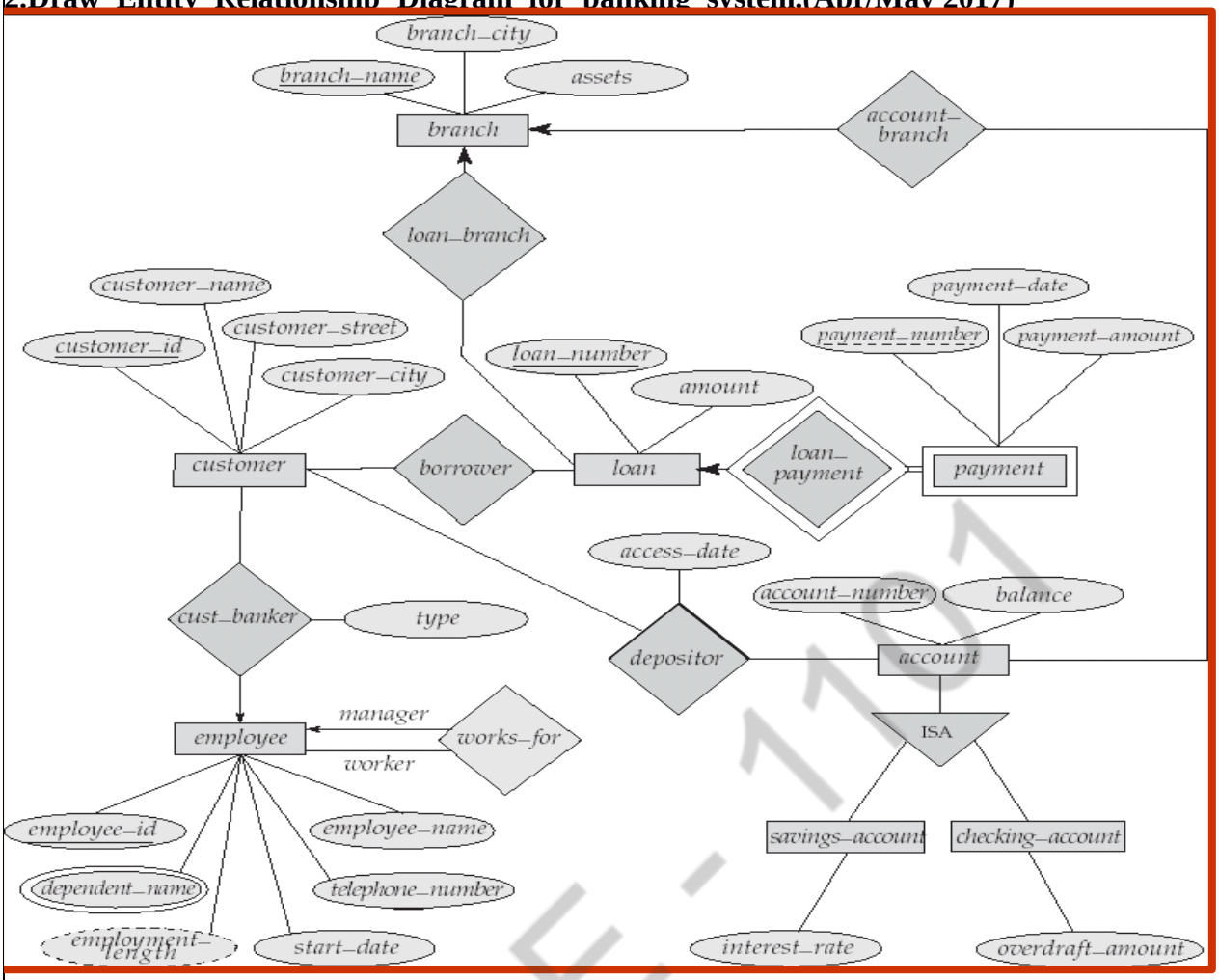
Partial Participation:

The participation of an entity set E in a relationship set R is said to be **partial** if only some entities in E participate in relationships in R.

Example:

An individual can be a bank customer whether or not the person has a loan with the bank. Hence it is possible that only some of the customer entities are related to the loan entity set through the borrower relationship. Hence the participation of customer in the borrower relationship set is partial.

2.Draw Entity Relationship Diagram for banking system.(Apr/May 2017)



3.Explain Normalization Forms 1NF,2NF,3NF,BCNF,4NF and 5NF.(May/JUNE 2016)

Functional Dependency

Definition. A **functional dependency**, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a *constraint* on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$, they must also have $t_1[Y] = t_2[Y]$.

The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations.**Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties .It reduce redundancy by using primary key constraints.

FIRST NORMAL FORM

i)Disallows composite attributes,multivalued attributes,

ii)nested relations; attributes whose values for an *individual tuple* are non-atomic

(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations

(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 10.8
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, AliciaJ.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Figure 10.9
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Example extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

SECOND NORMAL FORM

Prime attribute: An attribute that is member of the primary key K

Full functional dependency: a FD Y -> Z where removal of any attribute from Y means the FD does not hold any more

Partial functional dependency: a FD Y->Z where removal of any attribute from Y means the FD exists.

Examples:

{SSN, PNUMBER} -> HOURS is a full FD since neither SSN -> HOURS nor PNUMBER -> HOURS hold

{SSN, PNUMBER} -> ENAME is not a full FD (it is called a partial dependency) since SSN -> ENAME also holds

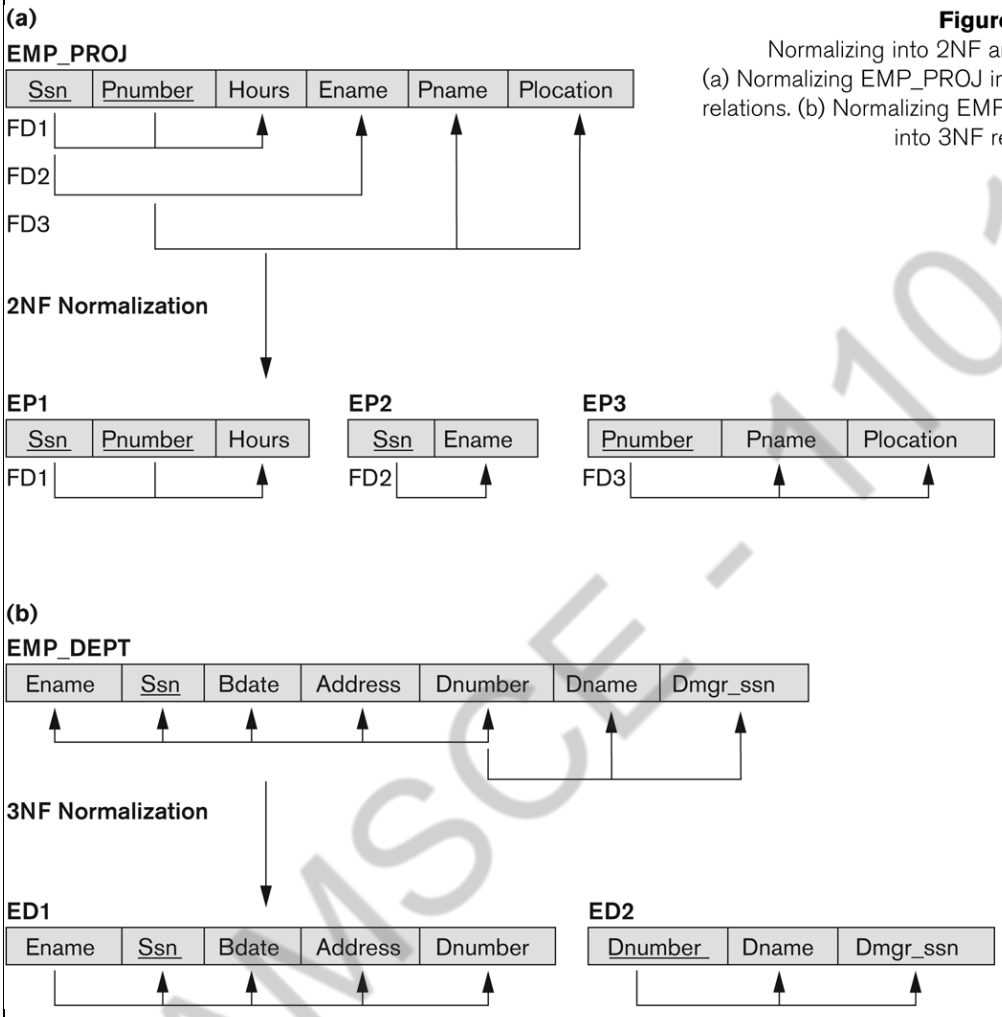


Figure 10.10
Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

THIRD NORMAL FORM

Transitive functional dependency: a FD X -> Z that can be derived from two FDs X -> Y and Y -> Z

Examples:

SSN -> DMGRSSN is a **transitive** FD

Since SSN -> DNUMBER and DNUMBER -> DMGRSSN hold

SSN -> ENAME is **non-transitive**

Since there is no set of attributes X where SSN -> X and X -> ENAME

A relation schema R is in **third normal form (3NF)** if whenever a FD X -> A holds in R, then either:

(a) X is a superkey of R, or

(b) A is a prime attribute of R

In X -> Y and Y -> Z, with X as the primary key, we consider this a problem only if Y is not a candidate key.

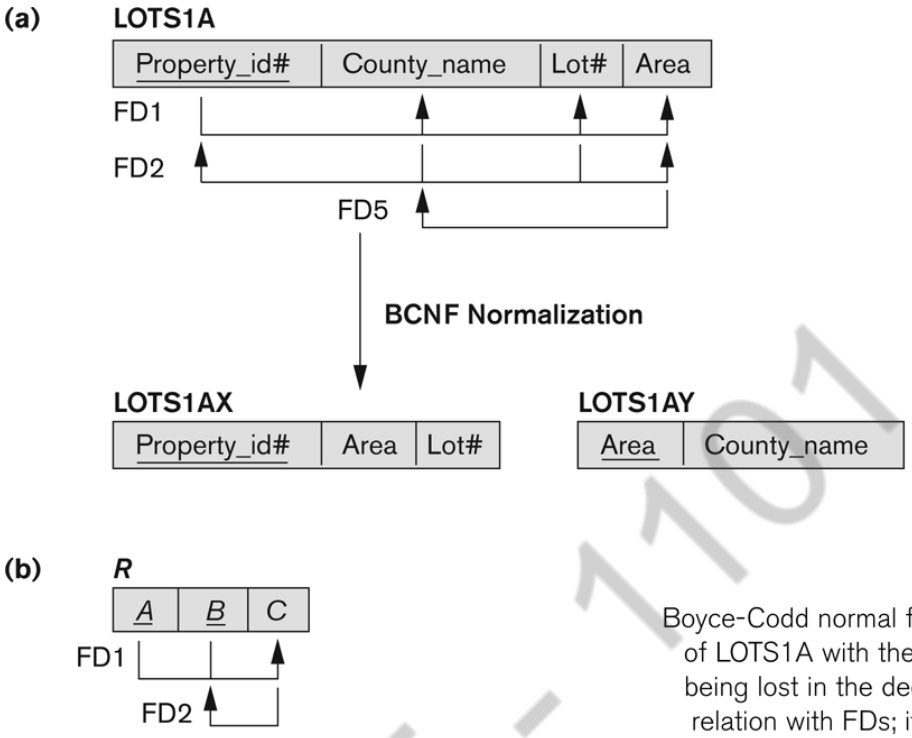
When Y is a candidate key, there is no problem with the transitive dependency .

E.g., Consider EMP (SSN, Emp#, Salary).

Here, SSN -> Emp# -> Salary and Emp# is a candidate key.

BOYCEE CODD NORMAL FORM

A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever an **FD X -> A** holds in R, then **X is a superkey** of R.X(Leftside attribute) no need to be prime attribute.



Figur
Boyce-Codd normal form. (a) BCNF norma
of LOTS1A with the functional depender
being lost in the decomposition. (b) A sc
relation with FDs; it is in 3NF, but not in

Definition. A relation schema R is in **BCNF** if whenever a *nontrivial* functional dependency $X \rightarrow A$ holds in R, then X is a superkey of R.

FOURTH NORMAL FORM

Definition. A multivalued dependency $X \twoheadrightarrow Y$ specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties,[15] where we use Z to denote $(R - (X \cup Y))$:[16]

- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.
- $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$.
- $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$.

Definition. A relation schema R is in **4NF** with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $X \twoheadrightarrow Y$ in F^{+17} X is a superkey for R.

FIFTH NORMAL FORM

Definition. A **join dependency** (**JD**), denoted by JD(R_1, R_2, ..., R_n), specified on relation schema R, specifies a constraint on the states r of R. The constraint states that every legal state r of R should have a nonadditive join decomposition into R_1, R_2, ..., R_n. Hence, for every such r we have

$* (\pi_{R_1}(r), \pi_{R_2}(r), ..., \pi_{R_n}(r)) = r$

Definition. A relation schema R is in **fifth normal form** (**5NF**) (or **project-join normal form** (**PJNF**)) with respect to a set F of functional, multivalued, and join dependencies if, for every nontrivial join dependency JD(R_1, R_2, ..., R_n) in F^+ (that is, implied by F),[18] every R_i is a superkey of R.

Agent	Company	Product_Name
Suneet	ABC	Nut
Raj	ABC	Bolt
Raj	ABC	Nut
Suneet	CDE	Bolt
Suneet	ABC	Bolt

P1

Agent	Company
Suneet	ABC
Suneet	CDE
Raj	ABC

P2

Agent	Product_Name
Suneet	Nut
Suneet	Bolt
Raj	Bolt
Raj	Nut

P3

Company	Product_Name
ABC	Nut
ABC	Bolt
CDE	Bolt

Agent	Company	Product_Name
Suneet	ABC	Nut
Suneet	ABC	Bolt
Suneet	CDE	Nut*
Suneet	CDE	Bolt
Raj	ABC	Bolt
Raj	ABC	Nut

The spurious row as asterisked. Now, if this result is joined with P3 over the column 'company 'product_name' the following table is obtained:

Agent	Company	Product_name
Suneet	ABC	NUT
Suneet	ABC	BOLT
Suneet	CDE	BOLT
Raj	ABC	BOLT
Raj	ABC	NUT

UNIT III TRANSACTIONS

UNIT-III/ PART-A

1.Give the reasons for allowing concurrency?

The reasons for allowing concurrency is if the transactions run serially, a short transaction may have to wait for a preceding long transaction to complete, which can lead to unpredictable delays in running a transaction. So concurrent execution reduces the unpredictable delays in running transactions.

2.What is average response time?

The average response time is that the average time for a transaction to be completed after it has been submitted.

3.What are the two types of serializability?

The two types of serializability is Conflict serializability, View serializability.

4.Differentiate strict two phase locking protocol and rigorous two phase locking protocol.*(May/June 16)*

- ✓ In strict two phases locking protocol all exclusive mode locks taken by a transaction is held until that transaction commits.
- ✓ Rigorous two phase locking protocol requires that all locks be held until the Transaction commits.

5.How the time stamps are implemented

- Use the value of the system clock as the time stamp. That is a transaction's time stamp is equal to the value of the clock when the transaction enters the system.
- Use a logical counter that is incremented after a new timestamp has been assigned; that is the time stamp is equal to the value of the counter.

6.What are the different modes of lock?

The modes of lock are:

- ✓ Shared
- ✓ Exclusive

7.What are the time stamps associated with each data item?

- W-timestamp (Q) denotes the largest time stamp if any transaction that executed WRITE (Q) successfully.
- ✓ R-timestamp (Q) denotes the largest time stamp if any transaction that executed READ (Q) successfully.

8.Define blocks?

The database system resides permanently on nonvolatile storage, and is partitioned into fixed- length storage units called blocks.

9.Define deadlock?

Neither of the transaction can ever proceed with its normal execution. This situation is called deadlock

10.Define the phases of two phase locking protocol

Growing phase: a transaction may obtain locks but not release any lock.

Shrinking phase: a transaction may release locks but may not obtain any new locks.

11.Define upgrade and downgrade?

It provides a mechanism for conversion from shared lock to exclusive lock is known as upgrade.

It provides a mechanism for conversion from exclusive lock to shared lock is known as downgrade.

12.What is a database graph?

The partial ordering implies that the set D may now be viewed as a directed acyclic graph, called a database graph.

13.What are uncommitted modifications?

The immediate-modification technique allows database modifications to be output to the database while the transaction is still in the active state. Data modifications written by active transactions are called uncommitted modifications.

14.What is meant by buffer blocks?

The blocks residing temporarily in main memory are referred to as buffer blocks.

15.Define shadow paging.

An alternative to log-based crash recovery technique is shadow paging. This technique needs fewer disk accesses than do the log-based methods.

16.Define page.

The database is partitioned into some number of fixed-length blocks, which are referred to as pages.

17.Explain current page table and shadow page table.

The key idea behind the shadow paging technique is to maintain two page tables during the life of the transaction: the current page table and the shadow page table. Both the page tables are identical when the transaction starts. The current page table may be changed when a transaction performs a write operation.

18.What is transaction?

Collections of operations that form a single logical unit of work are called transactions.

What are the drawbacks of shadow-paging technique?

- Commit Overhead
- Data fragmentation
- Garbage collection

19.What is meant by garbage collection.*(May/June 16)*

Garbage may be created also as a side effect of crashes. Periodically, it is necessary to find all the garbage pages and to add them to the list of free pages. This process is called garbage collection.

20.What are the properties of transaction? Or Write the ACID properties of Transaction.

(Nov/Dec 14) (Apr/May 15)(May/June 16) Atomicity

, Consistency, Isolation and Durability

21.What is recovery management component?

Ensuring durability is the responsibility of a software component of the base system called the recovery management component.

22.When is a transaction rolled back?

Any changes that the aborted transaction made to the database must be undone. Once the changes caused by an aborted transaction have been undone, then the transaction has been rolled back.

23.Give an example of two phase commit protocol. *(Nov/Dec 15)*

Client want all or nothing transactions and Transfer either happens or nothing at all.

24.What are the states of transaction?

The states of transaction are

Active

Partially committed

Failed

Aborted

Committed

Terminated

25.What is a shadow copy scheme?

It is simple, but efficient, scheme called the shadow copy schemes. It is based on making copies of the database called shadow copies that one transaction is active at a time. The scheme also assumes that the database is simply a file on disk.

26.What is serializability? How it is tested? ***(Nov/Dec 14,16)***

A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule.
Precedence graph is used to test the serializability

27.What is transaction?

Collections of operations that form a single logical unit of work are called transactions.

28.What are the drawbacks of shadow-paging technique?

- Commit Overhead
- Data fragmentation
- Garbage collection

29.What is meant by garbage collection.*(May/June 16)*

Garbage may be created also as a side effect of crashes. Periodically, it is necessary to find all the garbage pages and to add them to the list of free pages. This process is called garbage collection.

30.What are the properties of transaction? Or Write the ACID properties of Transaction.

(Nov/Dec 14) (Apr/May 15)(May/June 16) Atomicity
, Consistency, Isolation and Durability

31.What is recovery management component?

Ensuring durability is the responsibility of a software component of the base system called the recovery management component.

32.When is a transaction rolled back?

Any changes that the aborted transaction made to the database must be undone. Once the changes caused by an aborted transaction have been undone, then the transaction has been rolled back.

33.Give an example of two phase commit protocol. ***(Nov/Dec 15)***

Client want all or nothing transactions and Transfer either happens or nothing at all.

34.What are the states of transaction?

The states of transaction are

Active
Partially committed
Failed
Aborted
Committed
Terminated

35.What is a shadow copy scheme?

It is simple, but efficient, scheme called the shadow copy schemes. It is based on making copies of the database called shadow copies that one transaction is active at a time. The scheme also assumes that the database is simply a file on disk.

36What is serializability? How it is tested? *(Nov/Dec 14,16)*

A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule.

Precedence graph is used to test the serializability

x is to allow a transaction to access a data item only if it is currently holding a lock on that item.

UNIT-3

PART-B

1.Explain different types of concurrency control techniques

Types of Locks:

There are various modes in which a data item may be locked. Some of the modes are:

Shared: If a transaction Ti has obtained a shared-mode lock (denoted by S) on item Q, then Ti can read, but cannot write, Q.

Exclusive: If a transaction Ti has obtained an exclusive-mode lock (denoted by X) on item Q, then Ti can both read and write Q.

The compatibility relation between the two modes of locking appears in the below matrix.

	S	X
S	true	false
X	false	false

An element comp(A, B) of the matrix has the value true if and only if mode A is compatible with mode B. Note that shared mode is compatible with shared mode, but not with exclusive mode. At any time, several shared-mode locks can be held simultaneously (by different transactions) on a particular data item. A subsequent exclusive-mode lock request has to wait until the currently held shared-mode locks are released.

A transaction requests a shared lock on data item Q by executing the lock-S(Q) instruction. Similarly, a transaction requests an exclusive lock through the lock-X(Q) instruction. A transaction can unlock a data item Q by the unlock(Q) instruction.

Two phase locking protocol

A locking protocol is a set of rules indicating when a transaction may lock and unlock each of the data items. Locking protocols restrict the number of possible schedules.

One protocol that ensures serializability is the two-phase locking protocol. This protocol requires each transaction to issue lock and unlock requests in two phases:

Growing phase: A transaction may obtain locks, but may not release any lock.

Shrinking phase: A transaction may release locks, but may not obtain any new locks.

Intent Locking

Generally the unit for locking purposes is the individual tuple. In principle, however, there is no reason why locks should not be applied to larger or smaller units of data—for example, an entire relation, or even the entire database, or (going to the opposite extreme) a specific component within a specific tuple. The finer the granularity, the greater the concurrency. The fewer the locks that need to be set and tested the lower the overhead. For example, if a transaction has an X lock on an entire relation, there is no need to set X locks on individual tuples within that relation; on the other hand, no concurrent transaction will be able to obtain any locks on that relation, or on tuples within that relation, at all.

The X and S locks make sense for whole relations as well as for individual tuples. Other than this there are three additional kinds of locks, called intent locks, that also make sense for relations, but not for individual tuples:

- intent shared (IS) locks
- intent exclusive (IX) locks
- shared intent exclusive (SIX) locks

These new kinds of locks can *be* defined informally as follows.

Intent shared (IS): *T intends* to set S locks on individual tuples in *R,* in order to guarantee the stability of those tuples while they arc being processed

Intent exclusive (IX): *Same as IS, plus T* might update individual tuples in *R* and will therefore set X locks on those tuples.

Shared (S): *T* can tolerate concurrent readers, but not concurrent updaters, in *R (T* itself will not update *any tuples in R).*

Shared intent exclusive (SIX): Combines *S* and IX; that *is, T* can tolerate concurrent readers, but not concurrent updaters, in *R, plus T* might update individual tuples in *R* and will therefore set X locks on those tuples.

Exclusive (X): *T* cannot tolerate any concurrent access to *R at all (T* itself might or might not update individual tuples in R).

The Compatibility matrix for all lock modes.

Timestamp based protocol

Each transaction is issued a timestamp when it enters the system. If an old transaction T_i has time-stamp TS(T_i), a new transaction T_j is assigned time-stamp TS(T_j) such that TS(T_i) <TS(T_j).

The protocol manages concurrent execution such that the time-stamps determine the serializability order.

In order to assure such behavior, the protocol maintains for each data *Q* two timestamp values:

- **W-timestamp**(*Q*) is the largest time-stamp of any transaction that executed **write**(*Q*) successfully.
- **R-timestamp**(*Q*) is the largest time-stamp of any transaction that executed **read**(*Q*) successfully.

The timestamp ordering protocol ensures that any conflicting **read** and **write** operations are executed in timestamp order.

Suppose a transaction T_i issues a **read**(*Q*)

1. If TS(T_i) £ **W**-timestamp(*Q*), then T_i needs to read a value of *Q* that was already overwritten. Hence, the **read** operation is rejected, and T_i is rolled back.
2. If TS(T_i)³ **W**-timestamp(*Q*), then the **read** operation is executed, and R-timestamp(*Q*) is set to the maximum of R-timestamp(*Q*) and TS(T_i).

Suppose that transaction T_i issues **write**(*Q*).

If TS(T_i) < R-timestamp(*Q*), then the value of *Q* that T_i is producing was needed previously, and the system assumed that that value would never be produced. Hence, the **write** operation is rejected, and T_i is rolled back.

If TS(T_i) < W-timestamp(*Q*), then T_i is attempting to write an obsolete value of *Q*. Hence, this **write** operation is rejected, and T_i is rolled back.

Otherwise, the **write** operation is executed, and W-timestamp(*Q*) is set to TS(T_i).

Thomas write rule

When T_i attempts to write data item *Q*, if TS(T_i) < W-timestamp(*Q*), then T_i is attempting to write an obsolete value of {*Q*}. Hence, rather than rolling back T_i as the timestamp ordering protocol would have done, this {**write**} operation can be ignored.

Otherwise this protocol is the same as the timestamp ordering protocol.

Validation based protocol

Execution of transaction T_i is done in three phases

1. **Read and execution phase**: Transaction T_i writes only to temporary local variables
2. **Validation phase**: Transaction T_i performs a ``validation test'' to determine if local variables can be written without violating serializability.
3. **Write phase**: If T_i is validated, the updates are applied to the database;

otherwise, T_i is rolled back.

The three phases of concurrently executing transactions can be interleaved, but each transaction must go through the three phases in that order.

Also called as **optimistic concurrency control** since transaction executes fully in the hope that all will go well during validation

Each transaction T_i has 3 timestamps

- ✓ **Start**(T_i) : the time when T_i started its execution
- ✓ **Validation**(T_i): the time when T_i entered its validation phase
- ✓ **Finish**(T_i) : the time when T_i finished its write phase

Multiversion Timestamp ordering

Each data item Q has a sequence of versions $<Q_1, Q_2, ..., Q_m>$. Each version Q_k contains three data fields:

- ✓ **Content** -- the value of version Q_k.
- ✓ **W-timestamp**(Q_k) -- timestamp of the transaction that created (wrote) version Q_k
- ✓ **R-timestamp**(Q_k) -- largest timestamp of a transaction that successfully read version Q_k

when a transaction T_i creates a new version Q_k of Q, Q_k's W-timestamp and R-timestamp are initialized to TS(T_i).

R-timestamp of Q_k is updated whenever a transaction T_j reads Q_k, and TS(T_j) > R-timestamp(Q_k).

2.Explain different types of Serializability.

The database system must control concurrent execution of transactions, to ensure that the database state remains consistent. The concept of serializability examines how the database system can carry out this task. For this it is necessary to understand which schedules will ensure consistency, and which schedules will not.

Since transactions are programs, it is computationally difficult to determine exactly what operations a transaction performs and how operations of various transactions interact. The two main operations are: read and write.

The two forms of serializability are ***conflict serializability and view serializability.***

Conflict serializability:

Consider a schedule S in which there are two consecutive instructions Ii, and Ij, of transactions Ti and Tj, respectively. If Ii and Ij refer to different data items, then we can swap Ii, and Ij without affecting the results of any instruction in the schedule. However, if Ii and Ij refer to the same data item Q, then the order of the two steps may matter. Since transactions dealing with only read and write instructions, there are ***four cases*** that need to be considered to determine the order:

- Ii = read(Q), Ij = read(Q). The order of Ii and Ij does not matter, since the same value of Q is read by Ti and Tj, regardless of the order.
- Ii = read(Q), Ij = write(Q). If Ii comes before Ij, then Ti does not read the value of Q that is written by Tj in instruction Ij. If Ij comes before Ii then Ti reads the value of Q that is written by Tj. Thus, the order of Ii, and Ij matters.
- Ii = write(Q), Ij = read(Q). The order of Ii and Ij matters for reasons similar to those of the previous case.

- Ii = write(Q), Ij = write(Q). Since both instructions are write operations, the order of these instructions does not affect either Ti or Tj. However, the value obtained by the next read(Q) instruction of S is affected, since the result of only the latter of the two write instructions is preserved in the database. If there is no other write(Q) instruction after Ii and Ij in S, then the order of Ii and Ij directly affects the final value of Q in the database state that results from schedule S.

Thus, only in the case where both Ii and Ij are read instructions, the relative order of their execution does not matter.

Therefore Ii and Ij conflict if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.

View Serializability:

View serializability consider a form of equivalence that is less stringent than conflict equivalence, but that, like conflict equivalence, is based on only the read and write operations of transactions.

Consider two schedules S and S', where the same set of transactions participates in both schedules.

- ✓ For each data item Q, if transaction Ti reads the initial value of Q in schedule S, then transactions Ti must in schedule S', also read the initial value of Q.
- ✓ For each data item Q, if transaction Ti, executes read(Q) in schedule S, and if that value was produced by a write(Q) operation executed by transaction Tj, then the read(Q) operation of transaction Ti must, in schedule S', also read the value of Q that was produced by the same write(Q) operation of transaction Tj.
- ✓ For each data item Q the transaction, (if any) that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S'.

The concept of ***view equivalence leads to the concept of view serializability***. A schedule S is view serializable if it is view equivalent to a serial schedule.

As an illustration, suppose that we augment schedule 7 with transaction T6, and obtain ***schedule 9 as in below Figure.***

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Schedule 9 is view serializable.

3.Explain how to handle deadlock?

System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set.

Deadlock prevention protocols ensure that the system will *never* enter into a deadlock state. Some prevention strategies :

- Require that each transaction locks all its data items before it begins execution (predeclaration).
- Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order (graph-based protocol).

Following schemes use transaction timestamps for the sake of deadlock prevention alone.

wait-die scheme — non-preemptive

- older transaction may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead.
- a transaction may die several times before acquiring needed data item

wound-wait scheme — preemptive

- older transaction *wounds* (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones.
- may be fewer rollbacks than *wait-die* scheme.

Timeout-Based Schemes :

- a transaction waits for a lock only for a specified amount of time. After that, the wait times out and the transaction is rolled back.
- thus deadlocks are not possible
- simple to implement; but starvation is possible. Also difficult to determine good value of the timeout interval.

Deadlock Detection

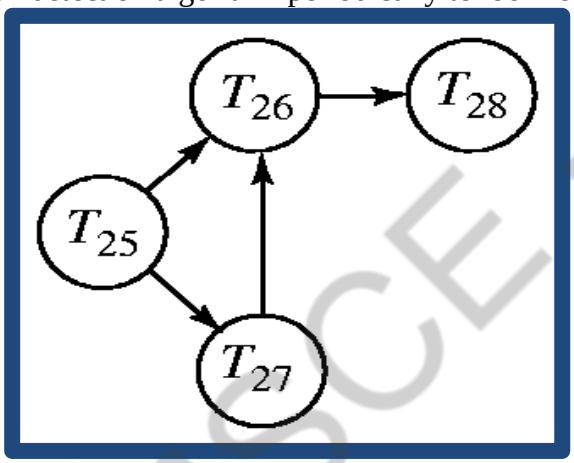
Deadlocks can be described as a *wait-for graph*, which consists of a pair $G = (V,E)$,

- V is a set of vertices (all the transactions in the system)
- E is a set of edges; each element is an ordered pair $T_i \rightarrow T_j$.

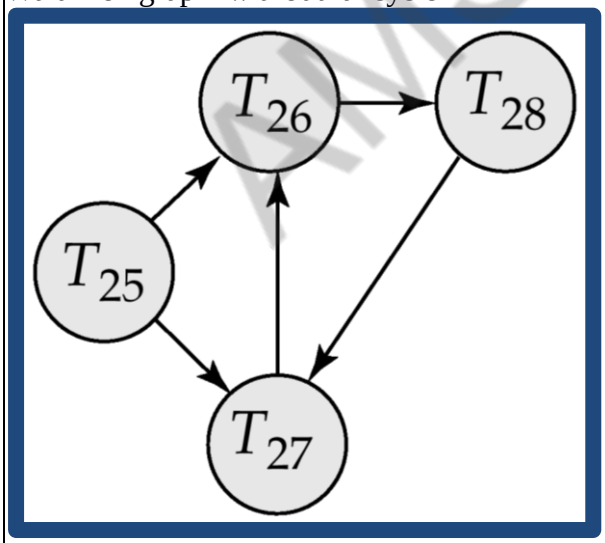
If $T_i \rightarrow T_j$ is in E, then there is a directed edge from T_i to T_j, implying that T_i is waiting for T_j to release a data item.

When T_i requests a data item currently being held by T_j, then the edge T_i T_j is inserted in the wait-for graph. This edge is removed only when T_j is no longer holding a data item needed by T_i.

The system is in a deadlock state if and only if the wait-for graph has a cycle. Must invoke a deadlock-detection algorithm periodically to look for cycles.



Wait –for-graph without a cycle



Wait –for-graph with a cycle

Deadlock Recovery

When deadlock is detected :

- Some transaction will have to rolled back (made a victim) to break deadlock. Select that transaction as victim that will incur minimum cost.
- Rollback -- determine how far to roll back transaction
  - Total rollback: Abort the transaction and then restart it.

- More effective to roll back transaction only as far as necessary to break deadlock.
- Starvation happens if same transaction is always chosen as victim. Include the number of rollbacks in the cost factor to avoid starvation

4.Explain in detail about ACID Properties.

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity,**C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

4) **Atomicity**− This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

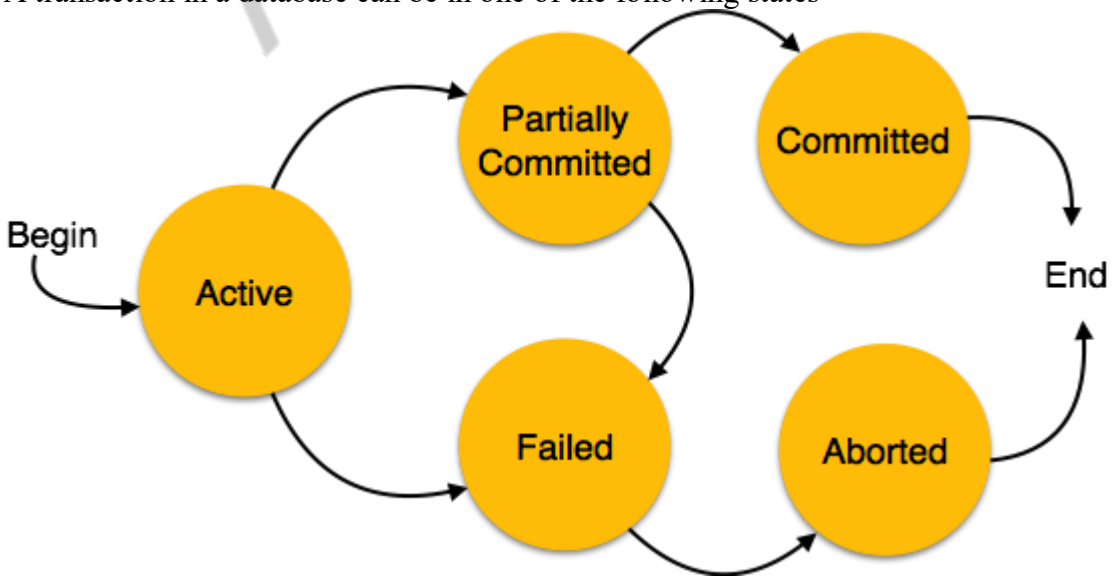
5) **Consistency**− The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

6) **Durability**− The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

7) **Isolation**− In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

5.Explain various States of Transactions.

A transaction in a database can be in one of the following states −



- ✓ **Active**− In this state, the transaction is being executed. This is the initial state of every transaction.
- ✓ **Partially Committed**− When a transaction executes its final operation, it is said to be in a partially committed state.
- ✓ **Failed**− A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
- ✓ **Aborted**− If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts −
    - Re-start the transaction
    - Kill the transaction

Committed− If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

6. **Explain Transaction Isolation Levels in DBMS(Apr/May 2017)**

As we know that, in order to maintain consistency in a database, it follows ACID properties. Among these four properties (Atomicity, Consistency, Isolation and Durability) Isolation determines how transaction integrity is visible to other users and systems. It means that a transaction should take place in a system in such a way that it is the only transaction that is accessing the resources in a database system.
Isolation levels define the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system. A transaction isolation level is defined by the following phenomena –

2. **Dirty Read –** A Dirty read is the situation when a transaction reads a data that has not yet been committed. For example, Let's say transaction 1 updates a row and leaves it uncommitted, meanwhile, Transaction 2 reads the updated row. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.

3. **Non Repeatable read –** Non Repeatable read occurs when a transaction reads same row twice, and get a different value each time. For example, suppose transaction T1 reads data. Due to concurrency, another transaction T2 updates the same data and commit, Now if transaction T1 rereads the same data, it will retrieve a different value.

4. **Phantom Read –** Phantom Read occurs when two same queries are executed, but the rows retrieved by the two, are different. For example, suppose transaction T1 retrieves a set of rows that satisfy some search criteria. Now, Transaction T2 generates some new rows that match the search criteria for transaction T1. If transaction T1 re-executes the statement that reads the rows, it gets a different set of rows this time.

Based on these phenomena, The SQL standard defines four isolation levels :

- **Read Uncommitted –** Read Uncommitted is the lowest isolation level. In this level, one transaction may read not yet committed changes made by other transaction, thereby allowing dirty reads. In this level, transactions are not isolated from each other.

- **Read Committed –** This isolation level guarantees that any data read is committed at the moment it is read. Thus it does not allows dirty read. The transaction holds a read or write lock on the current row, and thus prevent other transactions from reading, updating or deleting it.

- **Repeatable Read –** This is the most restrictive isolation level. The transaction holds read locks on all rows it references and writes locks on all rows it inserts, updates, or deletes. Since other transaction cannot read, update or delete these rows, consequently it avoids non-repeatable read.

- **Serializable –** This is the Highest isolation level. A *serializable* execution is guaranteed to be serializable. Serializable execution is defined to be an execution of operations in which concurrently executing transactions appears to be serially executing.

The Table is given below clearly depicts the relationship between isolation levels, read phenomena and locks :

Isolation Level	Dirty Read	Non Repeatable Read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

Anomaly Serializable is not the same as Serializable. That is, it is necessary, but not sufficient that a Serializable schedule should be free of all three phenomena types

Example

- SETTRANSACTION ISOLATIONLEVEL
- READ UNCOMMITTED
- BEGIN TRANSACTION MyTransaction
- BEGINTRY
- UPDATE Account SET Debit=100 WHERE Name='JohnCena'
- UPDATE ContactInformation SET Mobile='1234567890' WHERE Name='The Rock'

1. COMMIT TRANSACTION MyTransaction

- PRINT 'TRANSACTION SUCCESS'
- END TRY
- BEGIN CATCH
- ROLLBACK TRANSACTION MyTransaction
- PRINT 'TRANSACTION FAILED'
- END CATCH

UNIT IV IMPLEMENTATION TECHNIQUES

UNIT-IV / PART-A

1.	**What is B-Tree?** • A B-tree eliminates the redundant storage of search-key values. • It allows search key values to appear only once.
2.	**What is a B+-Tree index?** A B+-Tree index takes the form of a balanced tree in which every path from the root of the root of the root of the tree to a leaf of the tree is of the same length.
3.	**What is a hash index?** A hash index organizes the search keys, with their associated pointers, into a hash file structure
4.	**Define seek time.** The time for repositioning the arm is called the seek time and it increases with the distance that the arm is called the seek time.
5.	**Define rotational latency time.** The time spent waiting for the sector to be accessed to appear under the head is called the rotational latency time.
6.	**What is called mirroring?** The simplest approach to introducing redundancy is to duplicate every disk. This technique is called mirroring or shadowing.
7.	**What are the two main goals of parallelism?** • Load –balance multiple small accesses, so that the throughput of such accesses increases. • Parallelize large accesses so that the response time of large accesses is reduced.
8.	**What is an index?** An index is a structure that helps to locate desired records of a relation quickly, without examining all records
9.	**What are the factors to be taken into account when choosing a RAID level?** • Monetary cost of extra disk storage requirements. • Performance requirements in terms of number of I/O operations • Performance when a disk has failed and Performances during rebuild.
10.	**What are the types of storage devices?** Primary storage, Secondary storage, Tertiary storage, Volatile storage, Nonvolatile storage
11.	**What is called remapping of bad sectors?** If the controller detects that a sector is damaged when the disk is initially formatted, or when an attempt is made to write the sector, it can logically map the sector to a different physical location.

12.	**Define software and hardware RAID systems?*(May/June 16)*** RAID can be implemented with no change at the hardware level, using only software modification. Such RAID implementations are called software RAID systems and the systems with special hardware support are called hardware RAID systems.
13.	**Define hot swapping?** Hot swapping permits the removal of faulty disks and replaces it by new ones without turning power off. Hot swapping reduces the mean time to repair.
14.	**What are the ways in which the variable-length records arise in database systems?** Storage of multiple record types in a file, Record types that allow variable lengths for one or more fields, Record types that allow repeating fields.
15.	**What are the two types of blocks in the fixed –length representation? Define them.** Anchor block: Contains the first record of a chain. Overflow block: Contains the records other than those that are the first record of a chain.

16. What is hashing file organization?

In the hashing file organization, a hash function is computed on some attribute of each record. The result of the hash function specifies in which block of the file the record should be placed.

17. What are called index-sequential files?

The files that are ordered sequentially with a primary index on the search key are called index-sequential files.

18. Define Primary index and Secondary Index

It is in a sequentially ordered file, the index whose search key specifies the sequential order of the file. Also called clustering index. The search key of a primary index is usually but not necessarily the primary key. It is an index whose search key specifies an order different from the sequential order of the file. Also called non clustering index.

19. List out the mechanisms to avoid collision during hashing.*(Nov/Dec 16)*

In overflow chaining, the overflow buckets of a given bucket are chained together in a linked list.

Above scheme is called closed hashing. An alternative, called open hashing, which does not use overflow buckets, is not suitable for database applications.

20. What are the disadvantages of B-Tree over B+ Tree? *(Nov/Dec 16)*

Only small fraction of all search-key values are found early

✓ Non-leaf nodes are larger. Thus, B-Trees typically have greater depth than corresponding B+-Tree

Insertion and deletion more complicated than in B+-Trees

✓ Implementation is harder than B+-Trees.

✓ Not possible to sequentially scan a table by just looking at leafs.

21. What is called query processing?

Query processing refers to the range of activities involved in extracting data from a database.

22. What is called a query evaluation plan?

A sequence of primitive operations that can be used to evaluate be query is a query evaluation plan or a query execution plan.

23.	**Explain "Query optimization"?*(May/June 16)*** Query optimization refers to the process of finding the lowest cost method of evaluating a given query.
24.	**State the need for Query Optimization. *(Apr/May 15)*** The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans.

UNIT-IV / PART-B

1.Explain different types of Indexing methods.

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types –

- **Primary Index** − Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- **Clustering Index** − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Ordered Indexing is of two types −

- Dense Index
- Sparse Index

Dense Index

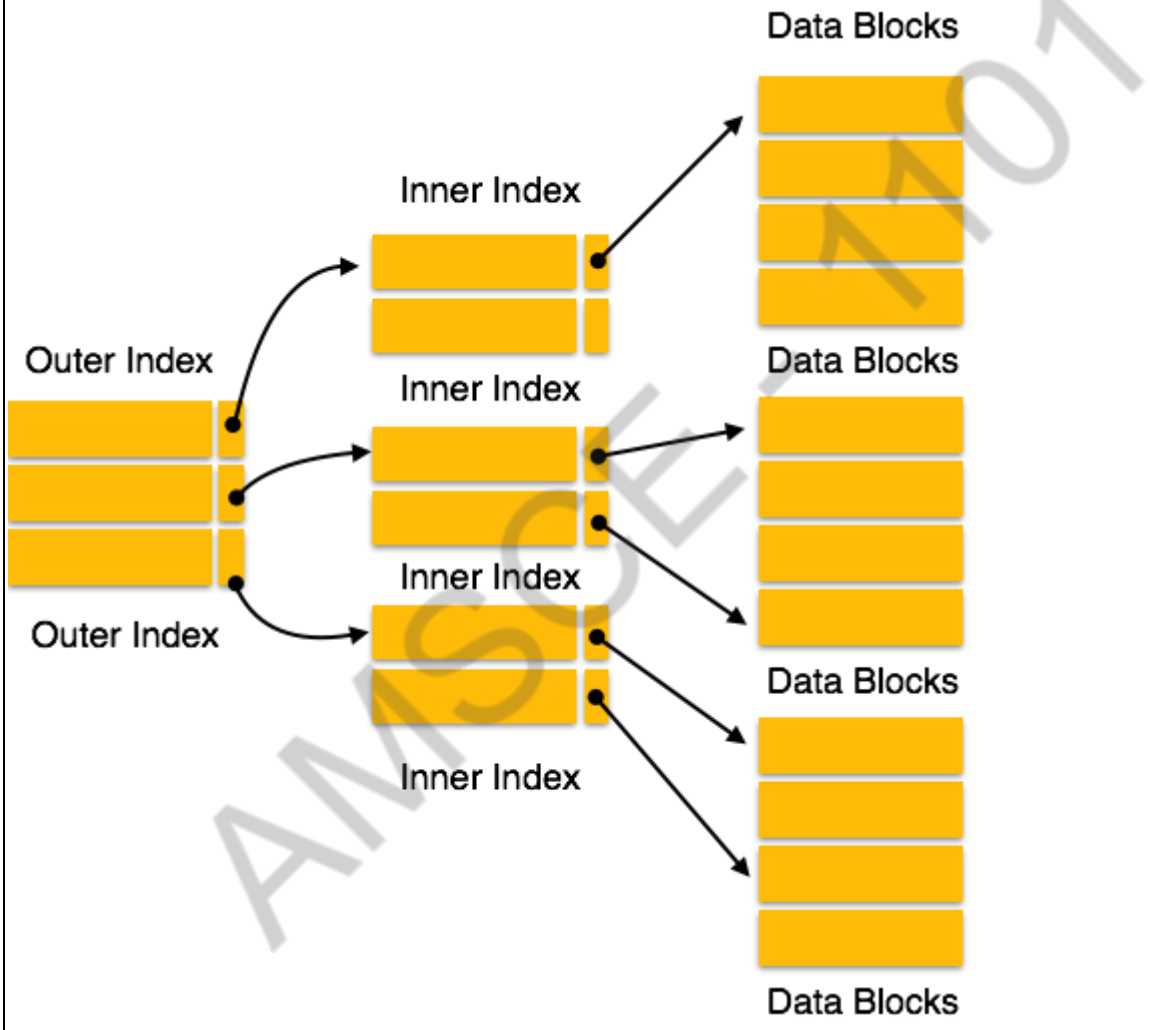
In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.

Sparse Index

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.

Multilevel Index

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

For a huge database structure, it can be almost next to impossible to search all the index values through all its level and then reach the destination data block to retrieve the desired data. Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure.

2. Explain different types of Hashing techniques.(or)Explain static and

dynamic hashing.

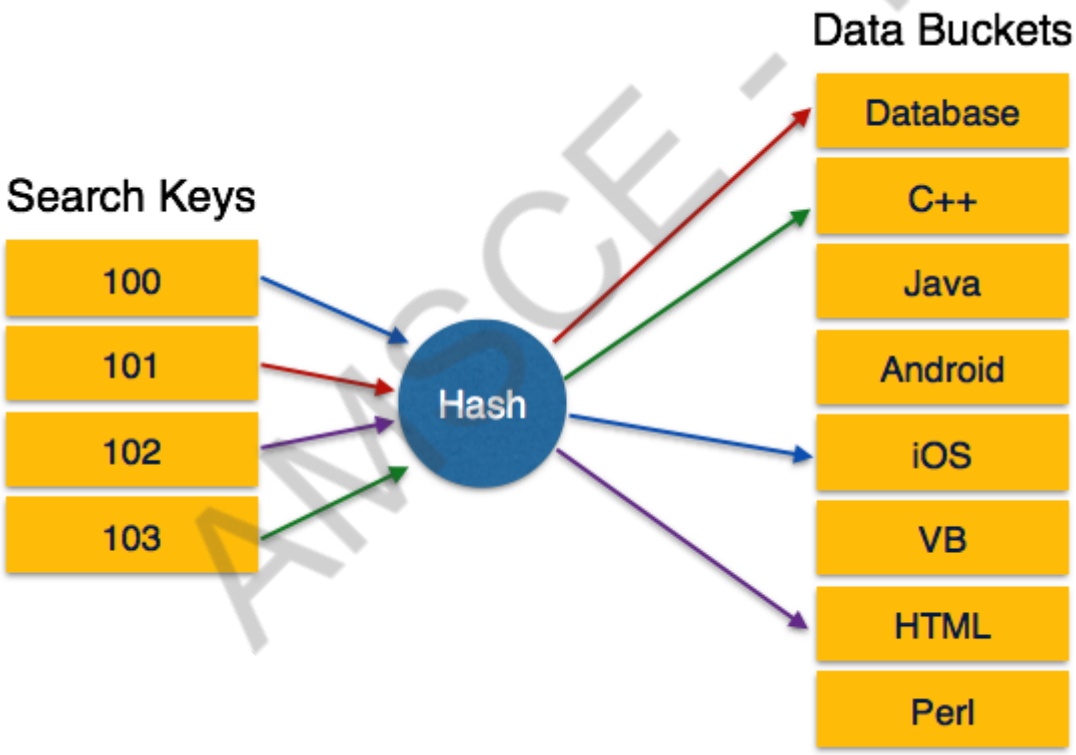
Hashing uses hash functions with search keys as parameters to generate the address of a data record.

Hash Organization

- **Bucket** − A hash file stores data in bucket format. Bucket is considered a unit of storage. A bucket typically stores one complete disk block, which in turn can store one or more records.
- **Hash Function** − A hash function, **h,** is a mapping function that maps all the set of search-keys **K** to the address where actual records are placed. It is a function from search keys to bucket addresses.

Static Hashing

In static hashing, when a search-key value is provided, the hash function always computes the same address. For example, if mod-4 hash function is used, then it shall generate only 5 values. The output address shall always be same for that function. The number of buckets provided remains unchanged at all times.



Operation

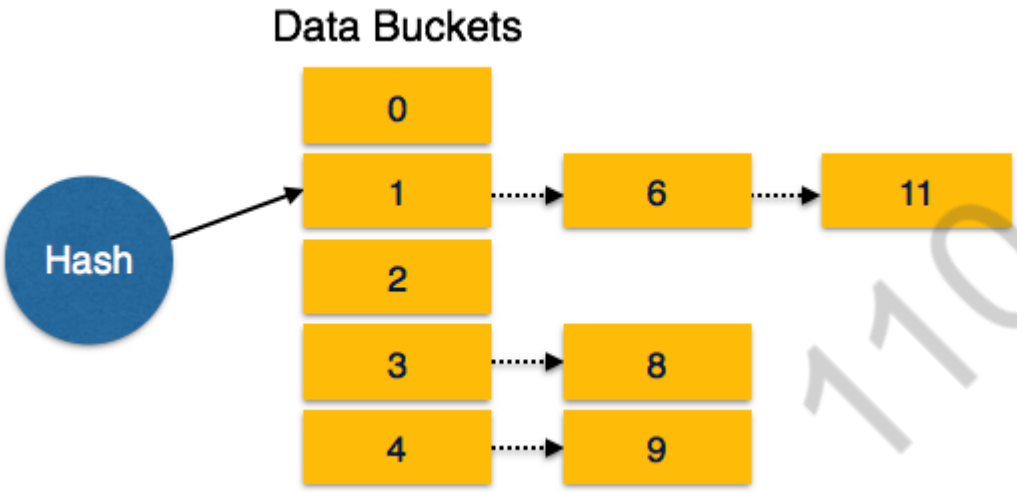
- **Insertion** − When a record is required to be entered using static hash, the hash function **h** computes the bucket address for search key **K**, where the record will be stored.
  Bucket address = h(K)
- **Search** − When a record needs to be retrieved, the same hash function can be used to retrieve the address of the bucket where the data is stored.

- **Delete** − This is simply a search followed by a deletion operation.

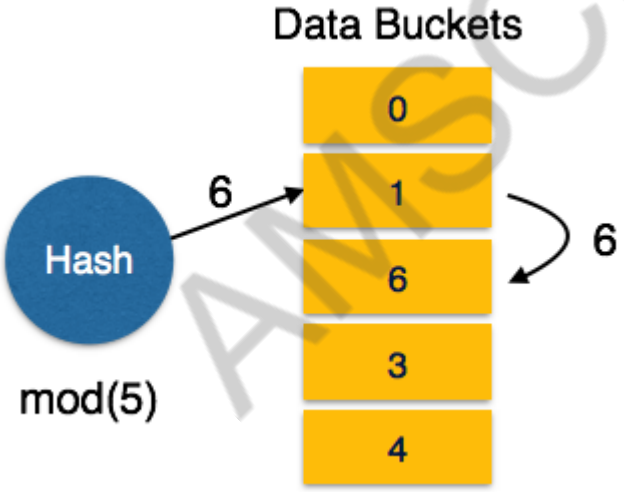
Bucket Overflow

The condition of bucket-overflow is known as **collision**. This is a fatal state for any static hash function. In this case, overflow chaining can be used.

- **Overflow Chaining** − When buckets are full, a new bucket is allocated for the same hash result and is linked after the previous one. This mechanism is called **Closed Hashing**.



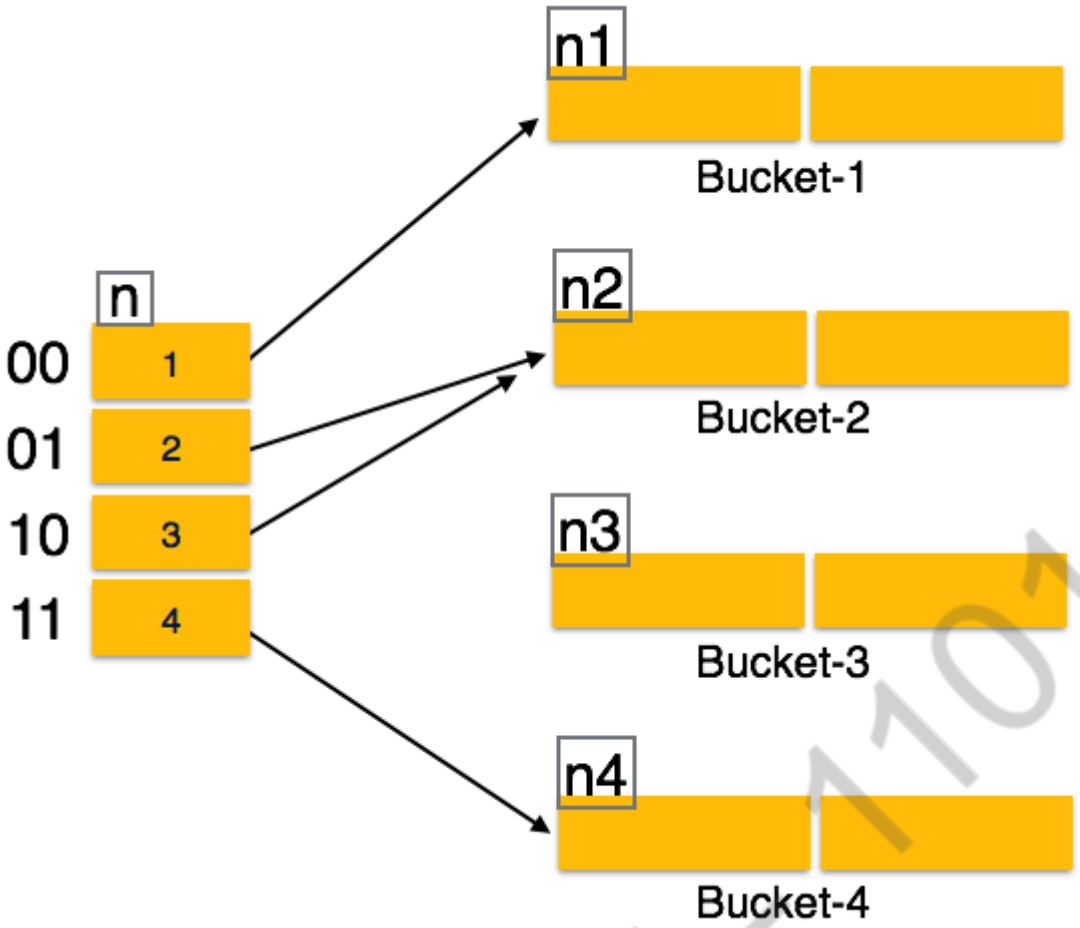
- **Linear Probing** − When a hash function generates an address at which data is already stored, the next free bucket is allocated to it. This mechanism is called **Open Hashing**.



Dynamic Hashing

The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks. Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand. Dynamic hashing is also known as **extended hashing**.

Hash function, in dynamic hashing, is made to produce a large number of values and only a few are used initially.



Organization

The prefix of an entire hash value is taken as a hash index. Only a portion of the hash value is used for computing bucket addresses. Every hash index has a depth value to signify how many bits are used for computing a hash function. These bits can address 2n buckets. When all these bits are consumed − that is, when all the buckets are full − then the depth value is increased linearly and twice the buckets are allocated.

Operation

- **Querying** − Look at the depth value of the hash index and use those bits to compute the bucket address.
- **Update** − Perform a query as above and update the data.
- **Deletion** − Perform a query to locate the desired data and delete the same.
- **Insertion** − Compute the address of the bucket

*)If the bucket is already full.

- Add more buckets.
- Add additional bits to the hash value.
- Re-compute the hash function.

*)Else

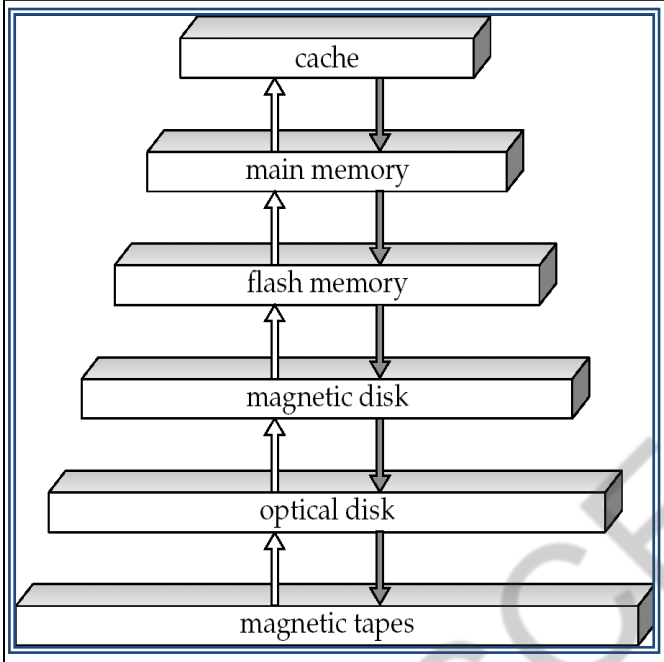
- Add data to the bucket,

*)If all the buckets are full, perform the remedies of static hashing.

Hashing is not favorable when the data is organized in some ordering and the queries require a range of data. When data is discrete and random, hash performs the best.

Hashing algorithms have high complexity than indexing. All hash operations are done in constant time.

3.Draw storage hierarchy of storage components.



4.Explain RAID Redundant Arrays of Independent Disks

*)Disk organization techniques that manage a large numbers of disks, providing a view of a single disk of

- high capacity and high speed by using multiple disks in parallel, and
- high reliability by storing data redundantly, so that data can be recovered even if a disk fails

***)Redundancy** – store extra information that can be used to rebuild information lost in a disk failure

E.g., **Mirroring** (or **shadowing**)

Duplicate every disk. Logical disk consists of two physical disks.

Two main goals of parallelism in a disk system:

1. Load balance multiple small accesses to increase throughput
2. Parallelize large accesses to reduce response time.

Improve transfer rate by striping data across multiple disks.

Bit-level striping – split the bits of each byte across multiple disks

But seek/access time worse than for a single disk

- Bit level striping is not used much any more

Block-level striping – with *n* disks, block *i* of a file goes to disk (i mod n) + 1

Requests for different blocks can run in parallel if the blocks reside on different disks

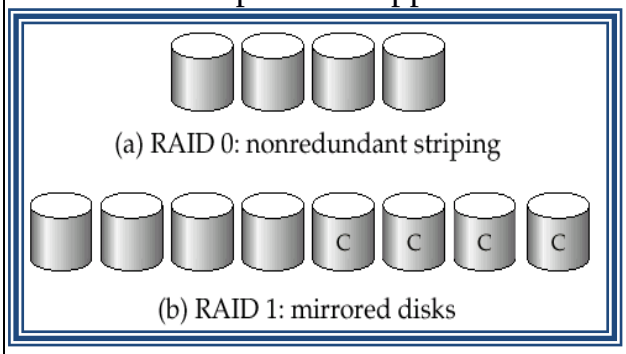
RAID Level 0: Block striping; non-redundant.

Used in high-performance applications where data lost is not critical.

RAID Level 1: Mirrored disks with block striping

Offers best write performance.

Popular for applications such as storing log files in a database system.

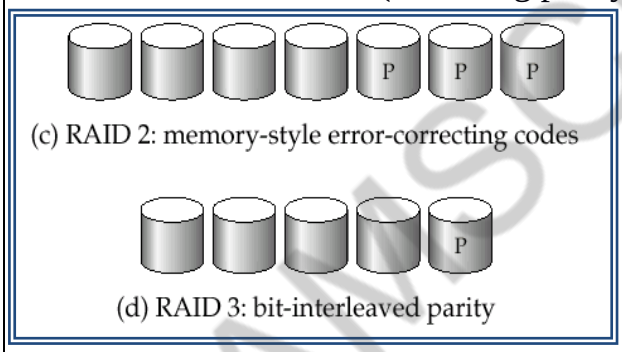


RAID Level 2: Memory-Style Error-Correcting-Codes (ECC) with bit striping.

RAID Level 3: Bit-Interleaved Parity

a single parity bit is enough for error correction, not just detection

- When writing data, corresponding parity bits must also be computed and written to a parity bit disk
- To recover data in a damaged disk, compute XOR of bits from other disks (including parity bit disk)

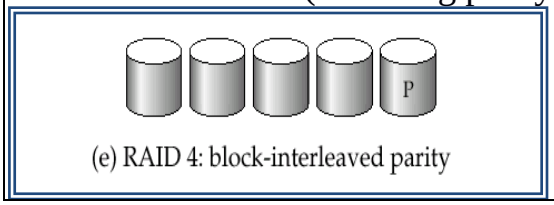


Faster data transfer than with a single disk, but fewer I/Os per second since every disk has to participate in every I/O.

RAID Level 4: Block-Interleaved Parity; uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from *N* other disks.

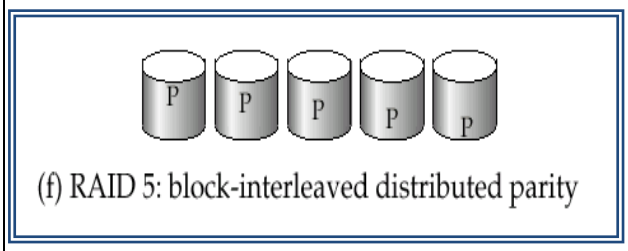
When writing data block, corresponding block of parity bits must also be computed and written to parity disk

To find value of a damaged block, compute XOR of bits from corresponding blocks (including parity block) from other disks.



RAID Level 5: Block-Interleaved Distributed Parity; partitions data and parity among all *N* + 1 disks, rather than storing data in *N* disks and parity in 1 disk.

- E.g., with 5 disks, parity block for *n*th set of blocks is stored on disk (*n mod* 5) + 1, with the data blocks stored on the other 4 disks.

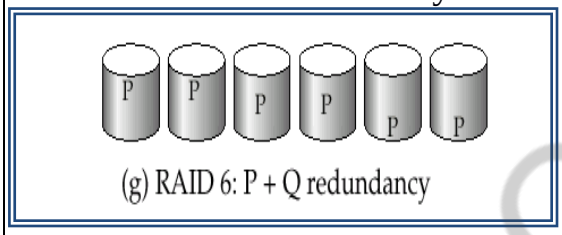


(f) RAID 5: block-interleaved distributed parity

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

RAID Level 6: P+Q Redundancy scheme; similar to Level 5, but stores extra redundant information to guard against multiple disk failures.

Better reliability than Level 5 at a higher cost; not used as widely.



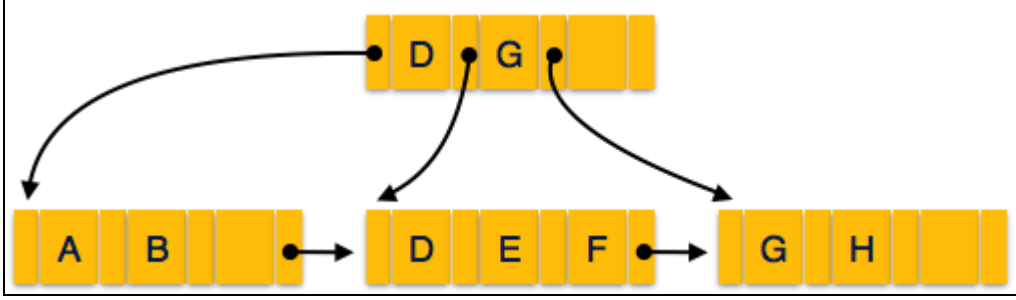
(g) RAID 6: P + Q redundancy

5.Explain B+tree indexing with example.

A B^+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B^+ tree denote actual data pointers. B^+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, the leaf nodes are linked using a link list; therefore, a B^+ tree can support random access as well as sequential access.

Structure of B^+ Tree

Every leaf node is at equal distance from the root node. A B^+ tree is of the order**n** where **n** is fixed for every B^+ tree.



Internal nodes −

- Internal (non-leaf) nodes contain at least ⌈n/2⌉ pointers, except the root node.
- At most, an internal node can contain **n** pointers.

Leaf nodes −

- Leaf nodes contain at least ⌈n/2⌉ record pointers and ⌈n/2⌉ key values.
- At most, a leaf node can contain **n** record pointers and **n** key values.
- Every leaf node contains one block pointer **P** to point to next leaf node and forms a linked list.

B^+ Tree Insertion

- B^+ trees are filled from bottom and each entry is done at the leaf node.
- If a leaf node overflows −
  - Split node into two parts.
  - Partition at $\mathbf{i = \lfloor (m+1)/2 \rfloor}$.
  - First **i** entries are stored in one node.
  - Rest of the entries (i+1 onwards) are moved to a new node.
  - $\mathbf{i}^{th}$ key is duplicated at the parent of the leaf.
- If a non-leaf node overflows −
  - Split node into two parts.
  - Partition the node at $\mathbf{i = \lceil (m+1)/2 \rceil}$.
  - Entries up to **i** are kept in one node.
  - Rest of the entries are moved to a new node.

B^+ Tree Deletion

- B^+ tree entries are deleted at the leaf nodes.
- The target entry is searched and deleted.
  - If it is an internal node, delete and replace with the entry from the left position.
- After deletion, underflow is tested,
  - If underflow occurs, distribute the entries from the nodes left to it.
- If distribution is not possible from left, then
  - Distribute from the nodes right to it.
- If distribution is not possible from left or from right, then
  - Merge the node with left and right to it.

6.**Explain BTREE indexing in detail.**

- B-tree is a specialized multiway tree designed especially for use on disk.
- B-Tree consists of a root node, branch nodes and leaf nodes containing the indexed field values in the ending (or leaf) nodes of the tree.
-

BTREE CHARACTERISTICS

- In a B-tree each node may contain a large number of keys
- B-tree is designed to branch out in a large number of directions and to contain a lot of keys in each node so that the height of the tree is relatively small
- Constraints that tree is always balanced
- Space wasted by deletion, if any, never becomes excessive
- Insert and deletions are simple processes

  Complicated only under special circumstances

  insertion into a node that is already full or a deletion from a node makes it lessWithin each node, $K_1 < K_2 < .. < K_{p-1}$
- Each node has at most p tree pointer
- Each node, except the root and leaf nodes, has at least ceil(p/2) tree pointers, The root node has at least two tree pointers unless it is the only node in the tree.
- All leaf nodes are at the same level. Leaf node have the same structure as internal nodes except that all of their tree pointer P_i are null.

  then half full

BTREE INSERTION

1) B-tree starts with a single root node (which is also a leaf node) at level 0.
2) Once the root node is full with p – 1 search key values and when attempt to insert another entry in the tree, the root node splits into two nodes at level 1.
3) Only the middle value is kept in the root node, and the rest of the values are split evenly between the other two nodes.
4) When a nonroot node is full and a new entry is inserted into it, that node is split into two nodes at the same level, and the middle entry is moved to the parent node along with two pointers to the new split nodes.
5) If the parent node is full, it is also split.
6) Splitting can propagate all the way to the root node, creating a new level if the root is split.

BTREE DELETION

1) If deletion of a value causes a node to be less than half full, it is combined with it neighboring nodes, and this can also propagate all the way to the root.

   Can reduce the number of tree levels.

7.Explain different types of file organization.

Heap File Organization

When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be placed anywhere in that memory area. It is the responsibility of the software to manage the records. Heap File does not support any ordering, sequencing, or indexing on its own.

Sequential File Organization

Every file record contains a data field (attribute) to uniquely identify that record. In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. Practically, it is not possible to store all the records sequentially in physical form.

Hash File Organization(Direct or Random)

Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.

Clustered File Organization

Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

8.Explain basic steps in query processing.

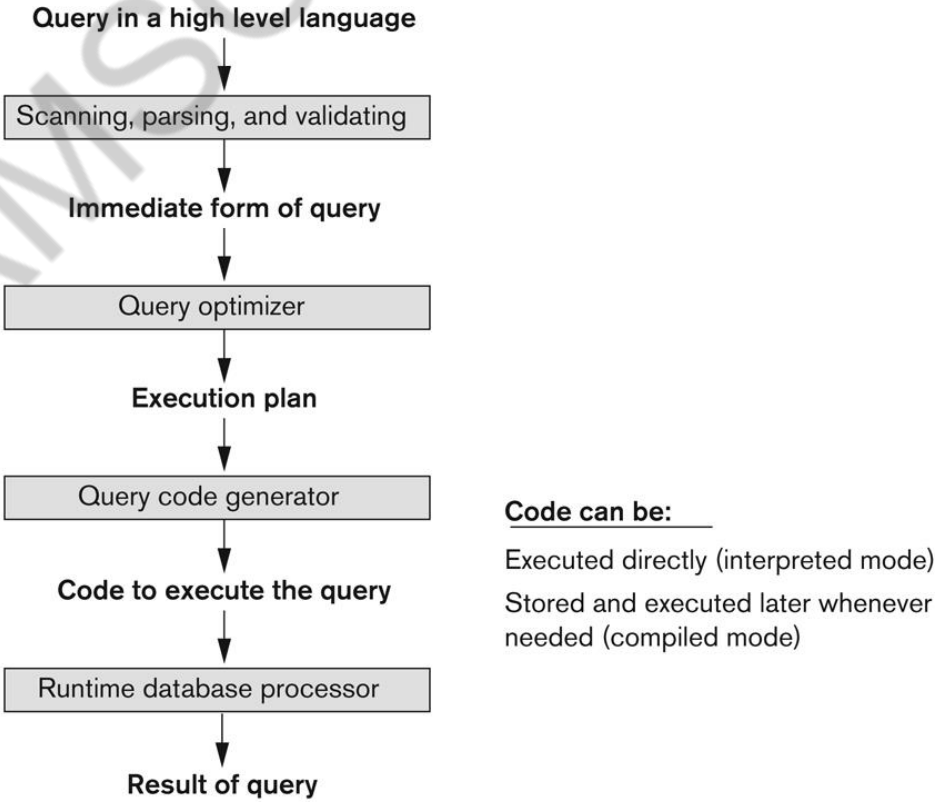


Figure 15.1
Typical steps when processing a high-level query.

- **Query optimization**: the process of choosing a suitable execution strategy for processing a query.

- Internal representation of a query

  Query Tree

- Parsing and translation

  *)translate the query into its internal form. This is then translated into relational algebra.
  *)Parser checks syntax, verifies relations

- Evaluation

*)The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

*)A relational algebra expression may have many equivalent expressions

E.g., $\sigma_{salary<75000}(\prod_{salary}(instructor))$ is equivalent to $\prod_{salary}(\sigma_{salary<75000}(instructor))$

- Each relational algebra operation can be evaluated using one of several different algorithms

  *)Correspondingly, a relational-algebra expression can be evaluated in many ways.

- Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**.

  *)E.g., can use an index on *salary* to find instructors with salary < 75000,
  *)or can perform complete relation scan and discard instructors with salary ≥ 75000

- **Query block:** the basic unit that can be translated into the algebraic operators and optimized.

- A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block.

- **Nested queries** within a query are identified as separate query blocks.
  - Aggregate operators in SQL must be included in the extended algebra.

UNIT V ADVANCED TOPICS

Distributed Databases: Architecture, Data Storage, Transaction Processing – Object-based Databases: Object Database Concepts, Object-Relational features, ODMG Object Model, ODL, OQL – XML Databases: XML Hierarchical Model, DTD, XML Schema, XQuery – Information Retrieval: IR Concepts, Retrieval Models, Queries in IR systems.

UNIT-V / PART-A

1.	**What is homogeneous distributed database and heterogeneous distributed database** A homogeneous distributed database has identical software and hardware running all databases instances, and may appear through a single interface as if it were a single database. A heterogeneous distributed database may have different hardware, operating systems, database management systems, and even data models for different databases.
2.	**Define Distributed Database Systems.** ***(Nov/Dec 16)*** Database spread over multiple machines (also referred to as sites or nodes).Network interconnects the machines. Database shared by users on multiple machines is called Distributed Database Systems
3.	**What are the types of Distributed Database** ✓ Homogenous distributed DB ✓ Heterogeneous distributed DB
4.	**Define fragmentation in Distributed Database** The system partitions the relation into several fragment and stores each fragment at different sites Two approaches : ✓ Horizontal fragmentation ✓ Vertical fragmentation

5.	**Define Database replication.** Database replication can be used on many database management systems, usually with a master/slave relationship between the original and the copies. The master logs the updates, which then ripple through to the slaves. The slave outputs a message stating that it has received the update successfully, thus allowing the sending of subsequent updates.
6.	**What is the advantage of OODB?** An integrated repository of information that is shared by multiple users, multiple products, multiple applications on multiple platforms.
7.	**What is Object database System?** An object database is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object-relational databases are a hybrid of both approaches.
8.	**What are the advantages of OODB?** An integrated repository of information that is shared by multiple users, multiple products, multiple applications on multiple platforms. It also solves the following problems: 1. The semantic gap: The real world and the Conceptual model is very similar. 2. Impedance mismatch: Programming languages and database systems must be interfaced to solve application problems. But the language style, data structures, of a programming language (such as C) and the DBMS (such as Oracle) are different. The OODB supports general purpose programming in the OODB framework. 3. New application requirements: Especially in OA, CAD, CAM, CASE, object-orientation is the most natural and most convenient.

9.	**How do you define types in object relational feature in oracle?** Oracle allows us to define types similar to the types of SQL. The syntax is CREATE TYPE t AS OBJECT (list of attributes and methods);
10.	**Define ODMG Object model?** The ODMG object model is the data model upon which the object definition langauge (ODL) and object query language (OQL) are based.
11.	**Define ODL.** ODL langauge is used to create object specifications: • classes and interfaces - Using the specific langauge bindings to specify how ODL • constructs can be mapped to constructs in specific programming • language, such as C++, SMALLTALK, and JAVA.
12.	**Define Information Retrieval.** It is an activity of obtaining information resources relevant to an information need from a collection of information resources.
13.	**Define Relevance Ranking.** ***(Nov/Dec 14)*** A system in which the search engine tries to determine the theme of a site that a link is coming from.
14.	**Can we have more than one constructor in a class? If yes, explain the need for such a situation.** ***(Nov/Dec 15)*** Yes, default constructor and constructor with parameter
15.	**Define XML Database.** An XML database is a data persistence software system that allows data to be stored in XML format. These data can then be queried, exported and serialized into the desired format. XML databases are usually associated with document-oriented databases.

16.	**Define OQL with syntax.** • Entry point to the database: needed for each query which can • be any named *persistent object*: ▪ the name of the extent of a class **class** Person (**extent** persons **key** ssn) { ... ... } **class** Faculty extends Person ENTRY POINTS (**extent** faculy) { ..... } **class** Department (**extent** departmet **key** dname){..... }
17.	**Define Crawling and indexing the web.** ***(Nov/Dec 14)*** Web Crawling is the process of search engines combing through web pages in order to properly index them. These "web crawlers" systematically crawl pages and look at the keywords contained on the page, the kind of content, all the links on the page, and then returns that information to the search engine's server for indexing. Then they follow all the hyperlinks on the website to get to other websites. When a search engine user enters a query, the search engine will go to its index and return the most relevant search results based on the keywords in the search term. Web crawling is an automated process and provides quick, up to date data.

18.	**How does the concept of an object in the object-oriented model differ from the concept of an entity in the entity-relationship model?*(Nov/Dec 16)*** An entity is simply a collection of variables or data items. An object is an encapsulation of data as well as the methods (code) to operate on the data. The data members of an object are directly visible only to its methods. The outside world can gain access to the object's data only by passing pre-defined messages to it and these messages are implemented by the methods.
19.	**Is XML Hierarchical?** XML documents have a hierarchical structure and can conceptually be interpreted as a tree structure, called an XML tree. XML documents must contain a root element (one that is the parent of all other elements). All elements in an XML document can contain sub elements, text and attributes.
20.	**What is DTD?** A document type definition (DTD) contains a set of rules that can be used to validate an XML file. After you have created a DTD, you can edit it manually, adding declarations that define elements, attributes, entities, and notations, and how they can be used for any XML files that reference the DTD file.
21.	**What is the use of XML Schema?** XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces.
22.	**What is Xpath and Xquery?** XPath can be used to navigate through elements and attributes in an XML document. XPath is a syntax for defining parts of an XML document. XPath uses path expressions to navigate in XML documents. XPath contains a library of standard functions. XPath is a major element in XSLT and in XQuery.
23.	**Define Keyword Queries.** Keyword-based queries are the simplest and most commonly used forms of IR queries: the user just enters keyword combinations to retrieve documents.

24.	**What are the Types of Queries in IR Systems** • Keyword Queries. Boolean Queries • Phrase Queries • Proximity Queries • Wildcard Queries • Natural Language Queries.
25.	**State the steps to create DTD.** Create a new DTD, complete the following steps: 1. Create a project to contain the DTD if needed. 2. In the workbench, click File > New > Other and select XML > DTD. Click Next. 3. Select the project or folder that will contain the DTD. 4. In the File name field, type the name of the DTD, for example MyDTD.dtd. The name of your DTD file must end with the extension .dtd 5. Click Next. 6. Optional: You can use a DTD template as the basis for your new DTD file. To do so, click the Use DTD Template check box, and select the template you want to use. 7. Click Finish.
	UNIT-V / PART-B
1	**1.Explain in detail about Distributed Databases**.(May/June 2016,2017) 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

In a homogeneous distributed database

- All sites have identical software

In a heterogeneous distributed database

- Different sites may use different schemas and softwareReplication
- 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.

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*.

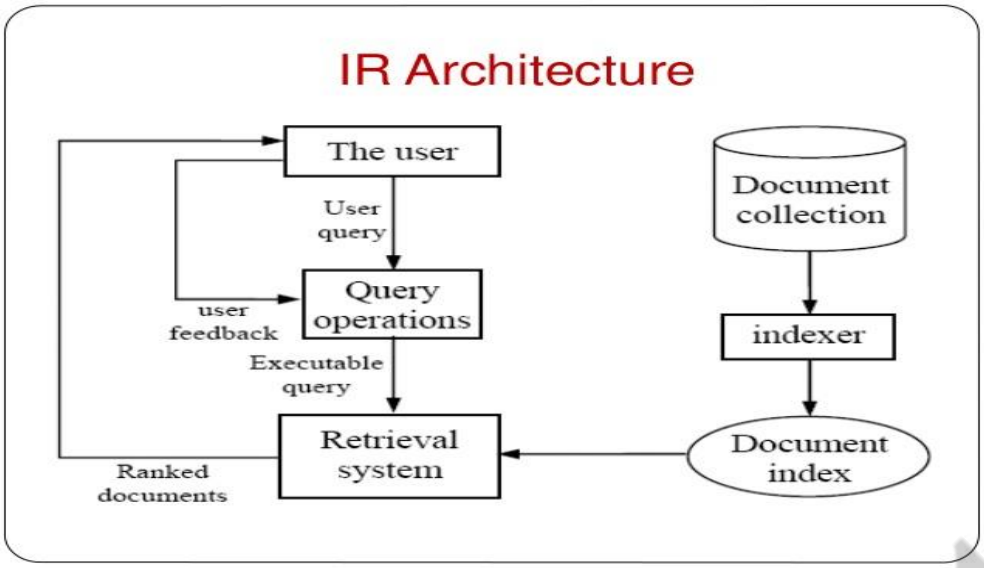
Division of relation r into fragments r_1, r_2, …, r_n which contain sufficient information to reconstruct relation r.
Horizontal fragmentation: each tuple of *r* is assigned to one or more fragments
Vertical fragmentation: the schema for relation *r* is split into several smaller schemas

- 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.

Example : relation account with following schema
Account-schema = (*branch-name, account-number, balance*)
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:

- Fragmentation transparency
- Replication transparency
- Location transparency

2.Explain about Information Retrieval.
Goal = find documents *relevant* to an information need from a large document set



The goal is to search large document collections (millions of documents) to retrieve small subsets relevant to the user's information need

Information retrieval (**IR**) systems use a simpler data model than database systems

Information organized as a collection of documents

Documents are unstructured, no schema

Information retrieval locates relevant documents, on the basis of user input such as keywords or example documents

e.g., find documents containing the words "database systems"

Can be used even on textual descriptions provided with non-textual data such as images

Web search engines are the most familiar example of IR systems

KEYWORD SEARCH

In **full text** retrieval, all the words in each document are considered to be keywords.

We use the word **term** to refer to the words in a document

Information-retrieval systems typically allow query expressions formed using keywords and the logical connectives *and*, *or*, and *not*

*And*s are implicit, even if not explicitly specified

Ranking of documents on the basis of estimated relevance to a query is critical

Relevance ranking is based on factors such as

Term frequency

Frequency of occurrence of query keyword in document

Inverse document frequency

How many documents the query keyword occurs in

Fewer è give more importance to keyword

Hyperlinks to documents

More links to a document è document is more important

Relavance Ranking

TF-IDF (Term frequency/Inverse Document frequency) ranking:

Let $n(d)$ = number of terms in the document d

$n(d, t)$ = number of occurrences of term t in the document d.

Relevance of a document d to a *term* t

The log factor is to avoid excessive weight to frequent terms

Relevance of document to *query Q*

Very common words such as "a", "an", "the", "it" etc are eliminated

Called **stop words**

Similarity based retrieval

retrieve documents similar to a given document

Similarity may be defined on the basis of common words

E.g. find *k* terms in A with highest *TF* (*d, t*) / *n* (*t*) and use these terms to find relevance of other documents.

Relevance feedback: Similarity can be used to refine answer set to keyword query

Synonyms

E.g. document: "motorcycle repair", query: "motorcycle maintenance"

need to realize that "maintenance" and "repair" are synonyms

System can extend query as "motorcycle *and* (repair *or* maintenance)"

Homonyms

E.g. "object" has different meanings as noun/verb

Can disambiguate meanings (to some extent) from the context

Examples are:

Internet search engines (Google, Yahoo! web search, etc.)

Digital library catalogues (MELVYL, GLADYS)

Some application areas within IR

Cross language retrieval

Speech/broadcast retrieval

Text categorization

Text summarization

Structured Document Element retrieval (XML)

3.Explain in detail about XML Databases.(May/June 2016,2017)

*)XML: Extensible Markup Language Defined by the WWW Consortium (W3C).It is derived from SGML (Standard Generalized Markup Language), but simpler to use than SGML .Documents have tags giving extra information about sections of the document.

*)Data interchange is critical in today's networked world

Examples:

- Banking: funds transfer
- Order processing (especially inter-company orders)
- Scientific data
  - Chemistry: ChemML
  - Genetics: BSML (Bio-Sequence Markup Language)

*)Paper flow of information between organizations is being replaced by electronic flow of information

)Each application area has its own set of standards for representing information

*)XML has become the basis for all new generation data interchange formats

*)Each XML based standard defines what are valid elements, using

i)XML type specification languages to specify the syntax

- DTD (Document Type Descriptors)
- XML Schema

ii)Plus textual descriptions of the semantics

*)XML allows new tags to be defined as required

However, this may be constrained by DTDs

STRUCTURE OF XML DATA

Tag: label for a section of data

Element: section of data beginning with <*tagname*> and ending with matching </*tagname*>

Elements must be properly nested

Proper nesting

- <course> … <title> …. </title> </course>

Improper nesting

- <course> … <title> …. </course> </title>

Formally: every start tag must have a unique matching end tag, that is in the context of the same parent element.

Every document must have a single top-level element

Example for nested Elements

```
<purchase_order>
<identifier> P-101 </identifier>
<purchaser> …. </purchaser>
<itemlist>
  <item>
    <identifier> RS1 </identifier>
    <description> Atom powered rocket sled </description>
    <quantity> 2 </quantity>
    <price> 199.95 </price>
  </item>
  <item>
    <identifier> SG2 </identifier>
    <description> Superb glue </description>
    <quantity> 1 </quantity>
    <unit-of-measure> liter </unit-of-measure>
    <price> 29.95 </price>
  </item>
 </itemlist>
</purchase_order>
```

*Elements can have **attributes**

```
<course course_id= "CS-101">
    <title> Intro. to Computer Science</title>
    <dept name> Comp. Sci. </dept name>
    <credits> 4 </credits>
</course>
```

*)Attributes are specified by *name=value* pairs inside the starting tag of an element

XML DOCUMENT SCHEMA

The type of an XML document can be specified using a DTD

*)DTD constraints structure of XML data

i)What elements can occur

ii)What attributes can/must an element have

iii)What subelements can/must occur inside each element, and how many times.

*)DTD does not constrain data types.All values represented as strings in XML

*)DTD syntax

```
<!ELEMENT element (subelements-specification) >
<!ATTLIST  element (attributes)  >
```

*)Subelements can be specified as

a)names of elements, or

b)#PCDATA (parsed character data), i.e., character strings

c)EMPTY (no subelements) or ANY (anything can be a subelement)

*)Example for Document type Definition(DTD)

```
<! ELEMENT department (dept_name  building, budget)>
<! ELEMENT dept_name (#PCDATA)>
<! ELEMENT budget (#PCDATA)>
```

*)Subelement specification may have regular expressions

```
<!ELEMENT university ( ( department | course | instructor | teaches )+)>
```

- Notation:
  - "|" - alternatives
  - "+" - 1 or more occurrences

- "*" - 0 or more occurrences

4.Explain in detail about distributed transaction processing.

Transaction may access data at several sites.

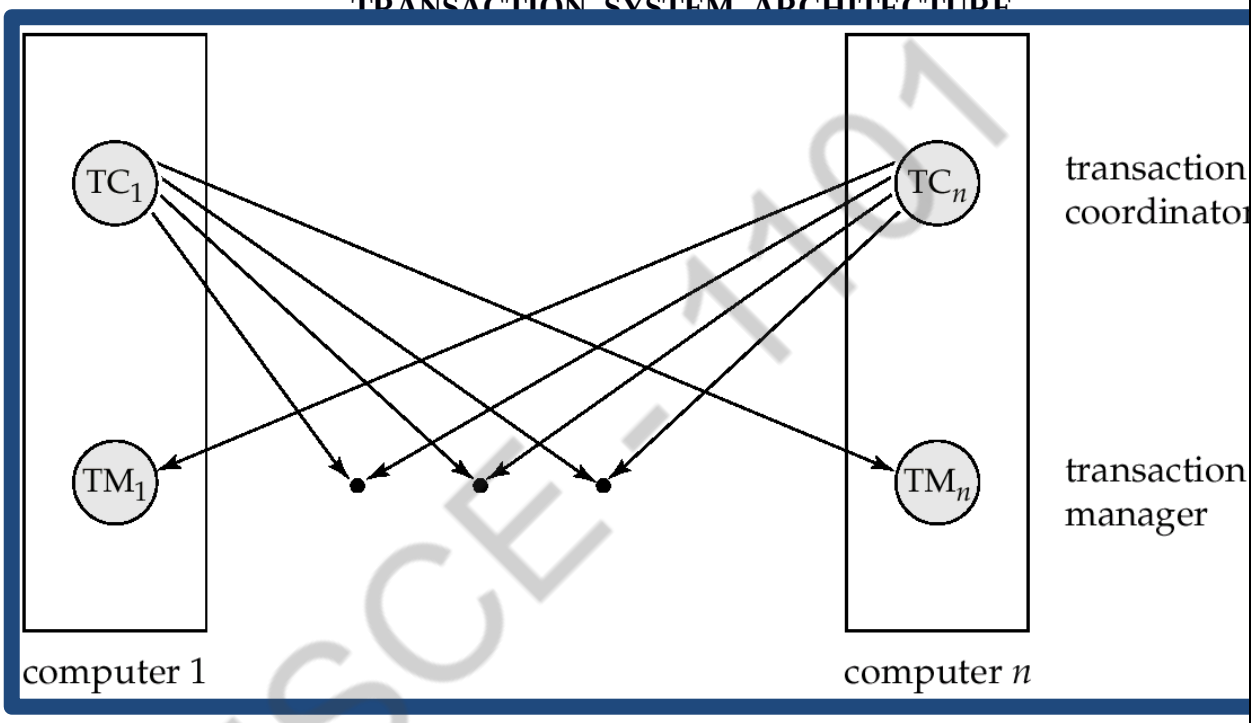
Each site has a local transaction manager responsible for:

- Maintaining a log for recovery purposes
- Participating in coordinating the concurrent execution of the transactions executing at that site.

Each site has a transaction coordinator, which is responsible for:

- Starting the execution of transactions that originate at the site.
- Distributing subtransactions at appropriate sites for execution.
- Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

TRANSACTION SYSTEM ARCHITECTURE



SYSTEM FAILURE MODES

Failures unique to distributed systems:

Failure of a site.

Loss of messages

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

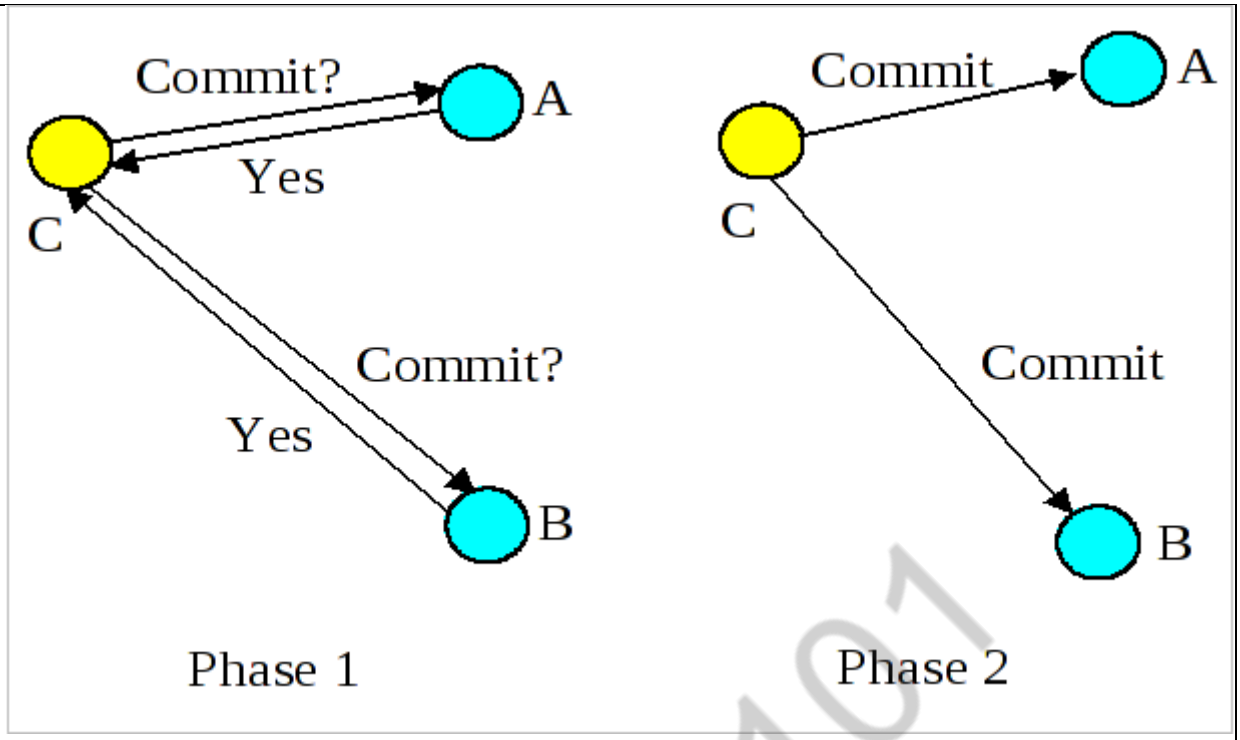
A network is said to be **partitioned** when it has been split into two or more subsystems that lack any connection between them

Note: a subsystem may consist of a single node

5.Explain about Two Phase Commit Protocol.(May/June 2016)

Conversation between transaction manager of all sites and coordinator

Basic commit/ rollback concept is called ***two-phase commit***: Two-phase commit is important whenever a given transaction can interact with several independent "resource managers," each managing its own set of recoverable resources and maintaining its own recovery log.



Assume for simplicity that the transaction has completed its database processing successfully, so that the system-wide instruction it issues is COMMIT, not ROLLBACK. On receiving that COMMIT request, the coordinator goes through the following two-phase process:

- ***Prepare****:* First, it instructs all resource managers to get ready to "go either way" on the transaction.

  In practice, this means that each participant in the process—that is, each resource manager involved—must force all log records for local resources used by the transaction out to its own physical log. Assuming the forced write is successful, the resource manager now replies "OK" to the coordinator; otherwise, it replies "Not OK".

- ***Commit****:* When the coordinator has received replies from all participants, it forces a record to its own physical log, recording its decision regarding the transaction. *If* all replies were "OK." that decision is "commit'."; if any reply was "Not OK," the decision is "rollback". Either way, the coordinator then informs each participant of its decision and *each participant must then commit or roll back the transaction locally, as instructed.*

If the system fails at some point during the foregoing process, the restart procedure will look for the decision record in the coordinator's log. If it finds it, then the two-phase commit process can pick up where it left off. If it does not find it, then it assumes that the action was "rollback." and the process can complete appropriately.

6.Explain about object oriented databases.

Motivation:

Permit non-atomic domains (atomic º indivisible)

Example of non-atomic domain: set of integers,or set of tuples

Allows more intuitive modeling for applications with complex data

Intuitive definition:

allow relations whenever we allow atomic (scalar) values — relations within relations

Retains mathematical foundation of relational model

Violates first normal form

Example: library information system

Each book has

title,

a set of authors,

Publisher, and

a set of keywords

Non-1NF relation *books*

title	*author-set*	*publisher*	*keyword-set*
		(*name, branch*)	
Compilers	{Smith, Jones}	(McGraw-Hill, New York)	{parsing, analysis}
Networks	{Jones, Frick}	(Oxford, London)	{Internet, Web}

STRUCTURE TYPES AND INHERITANCE

Structured types can be declared and used in SQL

```
create type Name as
(firstname      varchar(20),
 lastname       varchar(20))
      final
      create type Address as
(street         varchar(20),

 city    varchar(20),
 zipcode        varchar(20))
                not final
```

Note: **final** and **not final** indicate whether subtypes can be created

Structured types can be used to create tables with composite attributes

```
create table customer (
                name    Name,
                address         Address,
                dateOfBirth date)
```

Dot notation used to reference components: *name.firstname*

STRUCTURE TYPES

User-defined row types

```
create type CustomerType as (
        name Name,
        address Address,
        dateOfBirth date)
        not final
```

Can then create a table whose rows are a user-defined type

```
create table customer of CustomerType
```

METHODS

Can add a method declaration with a structured type.

```
        method ageOnDate (onDate date)
                returns interval year
```

Method body is given separately.

```
create instance method ageOnDate (onDate date)
                returns interval year
                for CustomerType
begin
                return onDate - self.dateOfBirth;
end
```

We can now find the age of each customer:

```
select name.lastname, ageOnDate (current_date)
from customer
```

INHERITANCE

Suppose that we have the following type definition for people:

```
                create type Person
           (name varchar(20),
         address varchar(20))
```

Using inheritance to define the student and teacher types

create type *Student*
under *Person*
(*degree* **varchar**(20),
department **varchar**(20))
create type *Teacher*
under *Person*
(*salary* **integer**,
department **varchar**(20))

Subtypes can redefine methods by using **overriding method** in place of **method** in the method declaration

MULTIPLE INHERITANCE

SQL:1999 and SQL:2003 do not support multiple inheritance

If our type system supports multiple inheritance, we can define a type for teaching assistant as follows:

create type *Teaching Assistant*
under *Student, Teacher*

To avoid a conflict between the two occurrences of *department* we can rename them

create type *Teaching Assistant*
under
Student **with** (*department* **as** *student_dept*),
Teacher **with** (*department* **as** *teacher_dept*)

ARRAYS AND MULTISET TYPES

Example of array and multiset declaration:

create type *Publisher* **as**
(*name* **varchar**(20),
branch **varchar**(20))
create type *Book* **as**
(*title* **varchar**(20),
author-array **varchar**(20) **array** [10],
pub-date **date**,
publisher *Publisher*,
keyword-set **varchar**(20) **multiset**)
create table *books* **of** *Book*

Similar to the nested relation books, but with array of authors instead of set

CREATION OF COLLECTION VALUES

Array construction

array [‘Silberschatz’,`Korth’,`Sudarshan’]

Multisets

multisetset [‘computer’, ‘database’, ‘SQL’]

To create a tuple of the type defined by the *books* relation: (‘Compilers’,
array[`Smith’,`Jones’],
Publisher (`McGraw-Hill’,`New York’),
multiset [`parsing’,`analysis’])

To insert the preceding tuple into the relation *books*

insert into *books*
values
(‘Compilers’, **array**[`Smith’,`Jones’],
Publisher (`McGraw-Hill’,`New York’),
multiset [`parsing’,`analysis’])

7.Explain object relational databases.

An object-relational database (ORD) is a database management system (DBMS) that's composed of both a relational database (RDBMS) and an object-oriented database (OODBMS). ORD supports the basic components of any object-oriented database model in its schemas and the query language used, such as objects, classes and inheritance.

An object-relational database may also be known as an object relational database management systems (ORDBMS).

One of the major goals of Object relational data model is to close the gap between relational databases and the object oriented practises frequently used in many programming languages such as C++, C#, Java etc.

Both Relational data models and Object oriented data models are very useful. But it was felt that they both were lacking in some characteristics and so work was started to build a model that was a combination of them both.

Advantages of Object Relational model

The advantages of the Object Relational model are:

Inheritance

The Object Relational data model allows its users to inherit objects, tables etc. so that they can extend their functionality. Inherited objects contains new attributes as well as the attributes that were inherited.

Complex Data Types

Complex data types can be formed using existing data types. This is useful in Object relational data model as complex data types allow better manipulation of the data.

Extensibility

The functionality of the system can be extended in Object relational data model. This can be achieved using complex data types as well as advanced concepts of object oriented model such as inheritance.

Disadvantages of Object Relational model

The object relational data model can get quite complicated and difficult to handle at times as it is a combination of the Object oriented data model and Relational data model and utilizes the functionalities of both of them.

we are performing *"table inheritance"* by doing this; since every student, teacher and parent is by definition also a "person" and we are guaranteed that we can work with all of those entities the same way by treating them as People if we want, or we can work with them using their specific attributes and relations.

Table inheritance::

create table People(PersonID int primary key, Name varchar ... etc ...)

create table Students(PersonID int primary key references People(PersonID), ...)

create table Teachers(PersonID int primary key references People(PersonID), ...)

create table Parents(PersonID int primary key references People(PersonID), ...)
This is because the unique constraint on the People table is just on the PersonID column, but we are trying to set up a foreign key constraint on the combination of PersonID/PersonTypeID. To handle this, we simply add an additional unique constraint to the People table, covering both PersonID and PersonTypeID:

```
create table People
(
  PersonID int primary key,
  PersonTypeID int references PersonType(PersonTypeID) not null,
  Name varchar(10) not null
)
create table Students
(
  PersonID int primary key,
  PersonTypeID as 1 persisted, -- student
  EnrollmentDate datetime,
  foreign key (PersonID, PersonTypeID) references People(PersonID, PersonTypeID)
)

create table Teachers
(
  PersonID int primary key,
  PersonTypeID as 2 persisted, -- teacher
  HireDate datetime,
  foreign key (PersonID, PersonTypeID) references People(PersonID, PersonTypeID)
)

create table Parents
(
  PersonID int primary key,
  PersonTypeID as 3 persisted, -- parents
  DifficultyScore int,
  foreign key (PersonID, PersonTypeID) references People(PersonID, PersonTypeID)
)
```