

What is Database system:-

UNIT-1

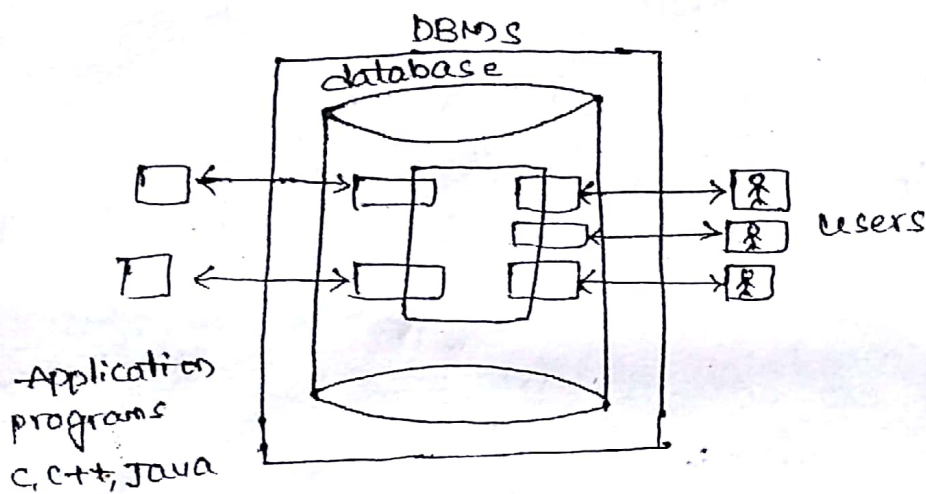
UTS

Database:- database is a collection of related data.

DBMS :- It is a sw which is used to manage the data base.

-Ex: Oracle, MySQL, SQL, PLSQL Etc..;

The Endusers can access the data with the help of Application programs and DBMS sw, the end users can access the database directly by using DBMS sw at the same time the Application programs cannot directly access the database.



-Ex: 1. Banking :-

customer information, accounts, loans and Banking transactions.

2. Airlines :-

Reservations, and scheduled information.

3. Universities :-

student information, Course Registrations, Grades.

4. Sales :-

customer, product, and purchase information.

Database characteristics :-

① Structured and described data :- It is a fundamental feature of database approach is that the database

System doesn't only contain the data. But, also contains the complete definition and description of these data that means structure, type, format, and relationship between the data. This kind of data is called "meta data". (Data about data).

② Separation of data and application:-

The application software doesn't need any knowledge about physical storage like encoding, format, storage place. It only communicates through the DBMS software.

③ Data Integrity:-

It maintains the quality and reliability of the data of a database system. It also provides protection from unauthorized users.

④ Transaction:-

A transaction is a bundle of actions which are done within a database to bring it from one consistent state to a new consistent state.

-e.g:- Transfer of amount from one account to another account.

⑤ Data persistence:-

In a DBMS maintain all the data as long as long period of time. The data is stored once that they can't be lost.

Components of Database:-

Data:- It is a very important component of the database system. The data acts as a bridge b/w the machine parts (hardware, software) and users which directly access it (or) access it through some application program.

Types of data:-

1) User data:- It consists of a table of data called relation where columns are called fields of attributes and rows are called records.

data
tion sh
meta

Metadata :- A description of the structure of data base is known as metadata (data about data).

Ex :- no. of tables and table names.

no. of fields and field names.

Application :- It stores the structure and format of queries, reports and other application components.

Hardware :- The hardware consists of secondary storage devices such as magnetic disk (hard disk, zip disk, floppy disk) optical disk (CD, ROM, magnetic tapes) These data can be store with the help of i/p devices (Keyboard, mouse, printer).

Software :- The software act as a bridge b/w the user and the database. The operations like insert, delete, update perform this DBMS SW (SQL).

Users :- users, are those persons who need the information from the data base. Those persons are :-

1. DBA (data base Administrator)

2. Data base designers, end users,

3. Application programmers.

What is database?

*) A database is a shared collection of related data used to support the activities of a particular organization. A database is a depository of data that can be accessed by various users.

Properties :-

* It is a representation of some aspects like real word information.

* Data base is logical, coherent and internally consistent.

* A Data base is design, built and populated with data for a specific purpose.

* Each data item is stored in a field.

* A combination of fields make up a table.

• Data base benefits :- (Why database).

* Self describing nature of a database system.

* Support for multiple views of data.

* Sharing of data and multiuser system.

* Control of data redundancy.

* Data sharing.

* Data integrity ~~Data Integrity~~

* Restriction of unauthorized actions.

* Data independence.

* Backup and recovery facilities.

• DBMS :-

• Data base management system is a collection of related data and set of programs which is used to store data and access the data into the database.

• DBMS is a general purpose s/w that facilitates the process of defining, constructing and manipulating databases for various applications.

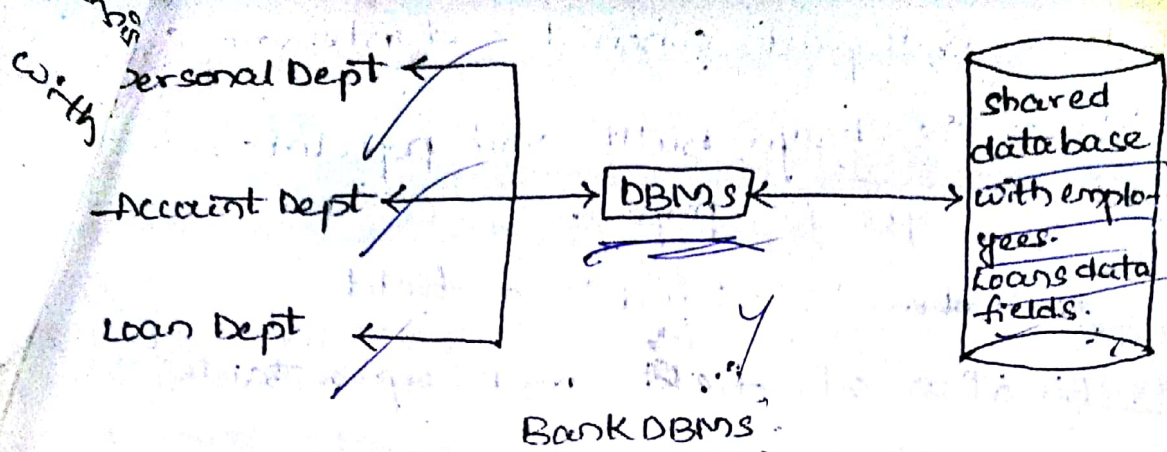
Goals :-

* manage large bodies of information.

* provide convenient and efficient way to store and access the information.

* To secure information against system failures.

* permit data to be shared among multiple users.



Advantages of DBMS:-

Data independence: meta data itself follows a layered architecture, so that when we change the data at one layer, it doesn't effect the data at another layer. This data is independent. This ability is called Data independence.

Efficient data access:-

DBMS utilizes a variety of techniques to store and retrieve data efficiently.

Data integrity and security:-

If the data is always accessed through the DBMS, it protects the data from unauthorized users.

Concurrent access and crash recovery:-

Data base allow multiple users to access the data concurrently. Any difficulties are occurred we get easily retrieve data by using backup and recovery facilities.

Data administration:-

Experienced person is needed to access and store the data to multiple users that person is called as database administration.

Reduce application data time:-

It takes the limited period of time to develop software applications.

Disadvantages of DBMS:-

Danger of a overkill :- Small and simple application for single user of a Database system is often not advisable.

Complexity :- A Database system creates additional complexity and requirements. Several users access the DBMS for data that's why it is quite costly and demanding.

Qualified person :- The professional of a Database system requires trained staff. Without qualified Database Administrator nothing will work for a long time.

Lower efficiency :- It contains lower efficiency. DBMS is a continuous multiuse slow which is often less efficient than specialized slow.

DBMS functions:-

Data definition :- The DBMS accepts data definitions (External, Conceptual, Internal, associated mapping) in source form and convert appropriate object form (tables).

Data manipulation :- The DBMS must be able to handle requests to retrieve, ^{obtain} ^{updated} (or) delete the existing data in the database.

Data optimization and execution :- The DML requests planned (or) unplanned must be processed by the optimizer.

Data security and integrity :-

If the data is always accessed through the DBMS slow and protects the data from unauthorized users.

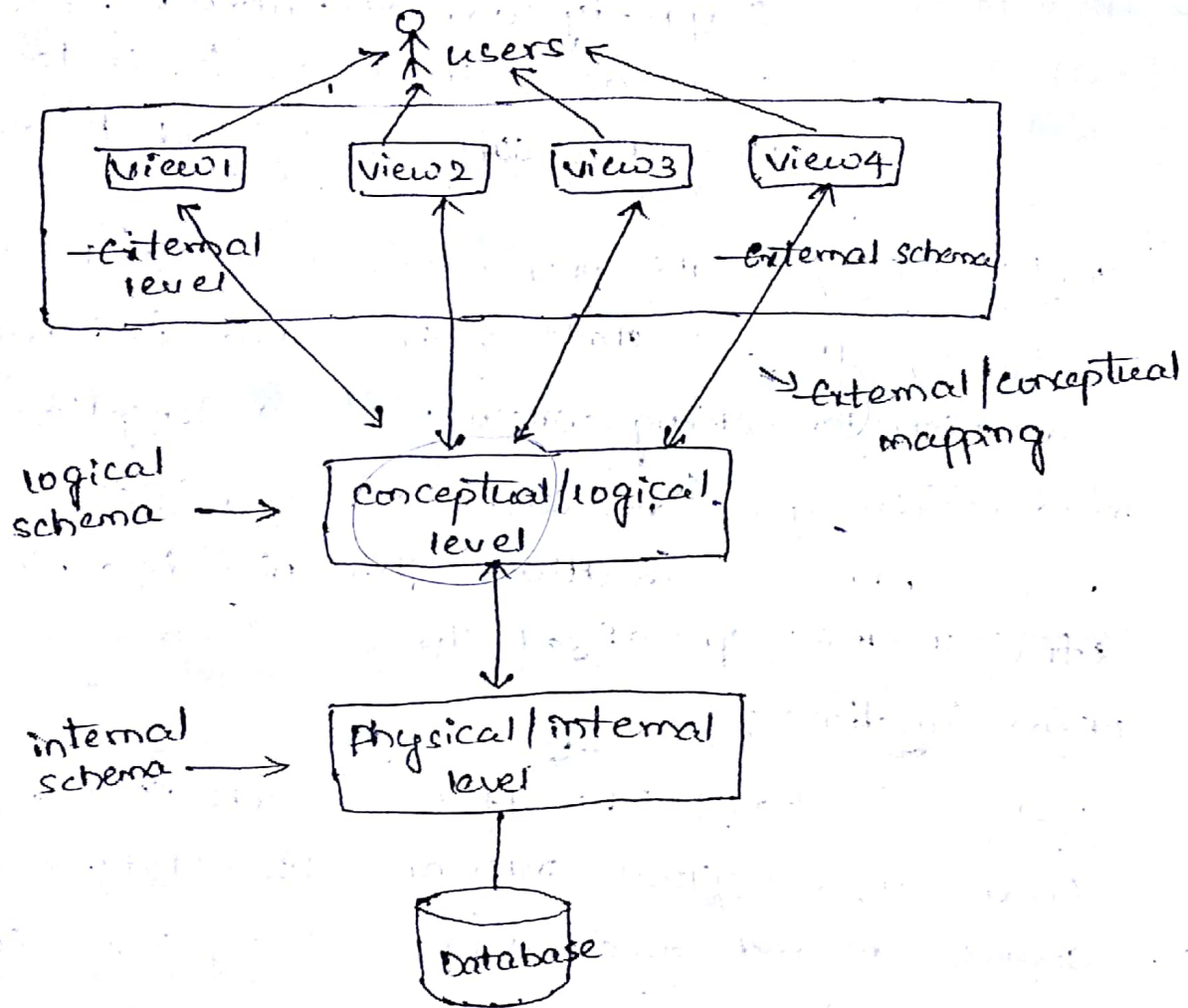
Data recovery and concurrency :-

Data base allow multiple users to access the data concurrently. Any difficulties are occurred we get

applicability. easily retrieve data by using backup and recovery facilities.

1. Data dictionary:—The DBMS provide a data dictionary function. The Dictionary can be regarded as a database (meta data).

2. Architecture of data base:—(3 level Architecture)



Mapping:—The above diagram shows the Architecture of database system.

- mapping is the process of transforming request, response b/w various database levels of architecture.
- mapping is not good for small database because it takes more time.
- In external/conceptual mapping, DBMS transforms a request on an external schema against the conceptual schema.

- In Conceptual / internal mapping to transform the request from conceptual to internal levels.

Physical level :- It describes the storage structure in a database.

• It is also known as internal level.

• This level is very close to physical storage of data.

• At lowest level, it is stored in the form of bits with physical address on the secondary storage devices.

• At highest level, it can be viewed in the form of files.

• Internal schema defines ^{various} stored data types. it uses physical database model.

Conceptual level :- It describes the structure of whole database for a group of users.

• It is also called as the 'data model'.

• Conceptual schema is representation of the entire content of the database.

• These schema contains all the information to build relevant external records.

• It hides the internal details of physical storage.

3. External level :-

• External level is related to the data which is viewed by individual end users.

• This level includes a no. of users views (or) external schemas.

• This level is closest to the user.

• External view describes the segment of

els. the database that is required for a particular user group and hides the rest of the database from that user group.

• Relational systems and others :-

The Relational system is a system in which

1. The data is collected by the user as tables
2. present we have to create new applications, the users/operators, collected data from old tables. i.e; derive new tables from olden tables.

Difference b/w relational and non-relational :-

• In a relational system the users cease the data as tables.

• In a relational system the user cease other data structures.

• The relational products begin from 1970's to 1980's.

ex:- oracle 9i from oracle corporation

DB₂ version from IBM.

• the object relational products begin from late 1980's-early 1990's

ex:- SQL products, DB₂, informix.

facultyname	salary	Telephone
Anand	45000	3340
Raja	33000	3372
Uma	18000	3342
Hari	28000	3341

select * faculty.

where salary > 3300

facultyname	salary	Telephone
Anand	45000	3340

• client server system:-

The purpose of database system is to support the development and execution of database applications.

The database system having simple two parts structure consisting of server also called Back end, and a set of clients also called as front end.

Server:- The server is nothing but DBMS s/w. It supports all of the basic functions of DBMS like (data definition, manipulation, security, integrity)

client:- The clients are various applications are run on the top of the DBMS (both user written application & building functions)

user written applications:-

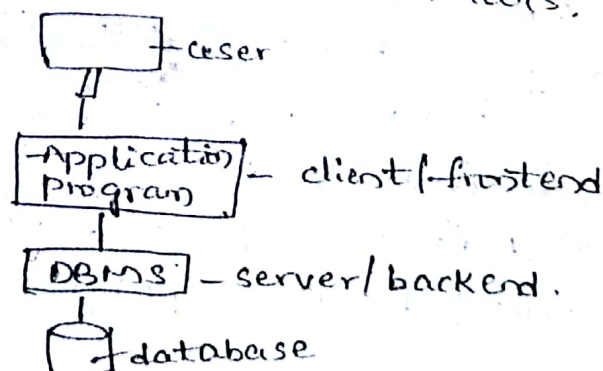
The user written applications are C, C++, Java.

vendor provided applications:-

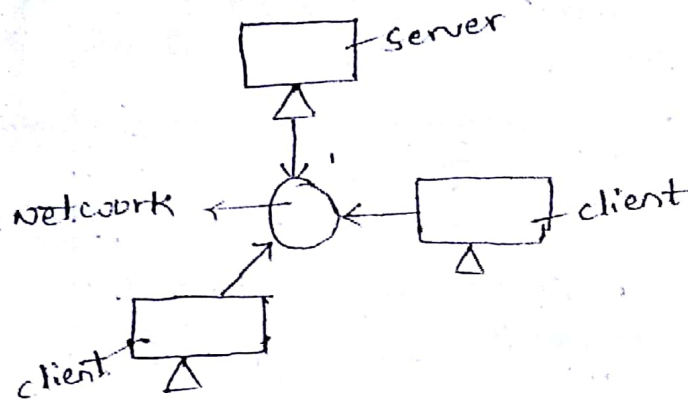
These are often called tools the purpose of the applications is creation and execution of their Applications.

Ex: Data mining and visualisation tools.

case tools



Process :- The client Applications and server can executed on the single same computer. The client server system also follows distributed processing. The distributed processing is the distinct machines can be connected into communication network.



single data processing task can share multiple (or) several machines in the network (parallel processing).

• DBA and its roles :-

The main reason for using DBMS is to have central control of the both data and the programs that access those data.

A person who has search central control over the system is called a database Administrator.

Roles :-

i) schema definition :- The DBA creates the original data base schema by writing set of definitions i.e; translated by the DDL compiler to a set of tables. i.e; stored permanently in the data dictionaries.

ii) storage structure and access method definition :- The programmers contain, make modifications either to a database schema (or) to the description of the physical storage organization by writing set of definition that is used by either the DDL compiler (or) the data storage and DDL compiler generate modifications to the appropriate internal system tables.

iii) Granting of Authorization for data access:-

* Granting different types of authorization allow the database administration to regulate which parts of the database various users can access. The authorization information is kept in a special system structure that is consulted by the database system whenever access to the data is attempted in the system.

iv) Monitoring performance and responding requirements:-

DBA is responsible for organizing the system. If we want change any requirement about storage we can easily changed through data independence concept.

v) Defining security & integrity constraints:-

Security and integrity constraints can be regarded as a part of the conceptual schema. The conceptual ddl must include facilities for specifying such constraints.

DBMS functions:-

- Data Definition
- Data manipulation
- Data security & integrity
- Data recovery and concurrency
- Data Dictionary.
- performance.

Data independence:-

A database system normally, contain layered architecture to store the data into the database. It follows the self describes nature meaning that metadata so that when we changed the data at one layer it doesn't effect the data at another level, this data is independent but mapped to each other. This is called data independence, & logical data independence.

logical schema → logical data independence

physical schema

→ physical data independence.

The data independence divided into two ways.

1. logical data independence.
2. physical data independence.

logical data independence :-

This level describes which data is stored in database and relationship among the data in database.

- ex: All entities, attributes and their relationship security and integrity information.
- * ability to change logical schema without schema changing the external-physical schemas.

physical data independence :-

This level describes how the data is stored in the database, it covers the data structure and the organization.

- * ability to change the physical data without effecting the logical data-external schema.

- ex: If you want to change different file organization storage devices, structure, indexes should be possible without effecting the conceptual-external schema.

10/1/19

UNIT - 2

Relation:- Representing the data in a relational model is called a relation.
 schema - structure of data

Relation schema:- It describes fields for the table, the schema specifies the relation name, name of each field, domain of each field

eg:- Student (SID: Int, SName: String, Branch: String)

Domain:- The domain of a database is the set of all allowable values (or) attributes of the database

eg:- Gender (Male, Female, unknown)

Degree:- The no. of fields (or) attributes in a relation is called degree of that relation.

Cardinality:- The no. of tuples in a relation, is called cardinality.

ex:- Student

ID	Name	Address
1.	Chenchu	562
2.	Kummu	563
3.	Mouni	593

Degree : 3
 cardinality : 3

Entity:- An Entity is an object which have some existence

* The existence is either physical (or) conceptual

eg:- person is an entity with physical existence

Job, course are entities with conceptual existence

Attribute:- The property of an entity is called attribute

eg:- Student is an entity with attributes SID, Sname, branch

Types of Attributes:-

1. Simple / Atomic attribute :- An attribute which is not divisible is called simple attribute.

eg:- Age of person Entity.

2. Composite Attribute:- An attribute which can be divisible into parts is called composite attribute.

eg:- Name & Address of person entity.

3. Single valued attribute:- An attribute having single value eg:- Age, DOB, ID

4. Multi valued attribute:- Attribute having multiple values eg:- phone number, email ID.

5. Stored Attribute:- An attribute which is used for deriving a new attribute.

eg:- DOB

6. Derived Attribute:- An attribute which is derived from another attribute.

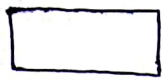
eg:- Age of person entity.

Relationship:- The association among the entity is called Relationship

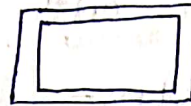
Relationship Set:- The Collection of Relationships is called Relationship Set.

* A Relationship can be defined with an attribute is called descriptive attribute.

E-R Diagrams:- The following symbols are used in database design.



→ Entity



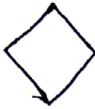
→ weak entity



→ Attribute



→ Multi value attribute



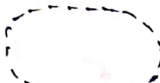
→ Relationship



→ weak relationship



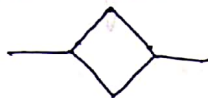
→ primary key of an attribute



→ derived attribute



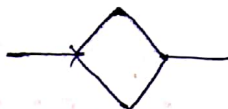
→ attribute of weak entity



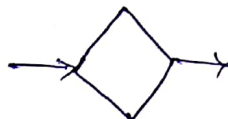
→ Many to many



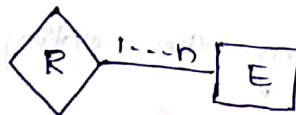
→ Many to one



→ one to many



→ one to one



→ cardinality limits



→ generalization

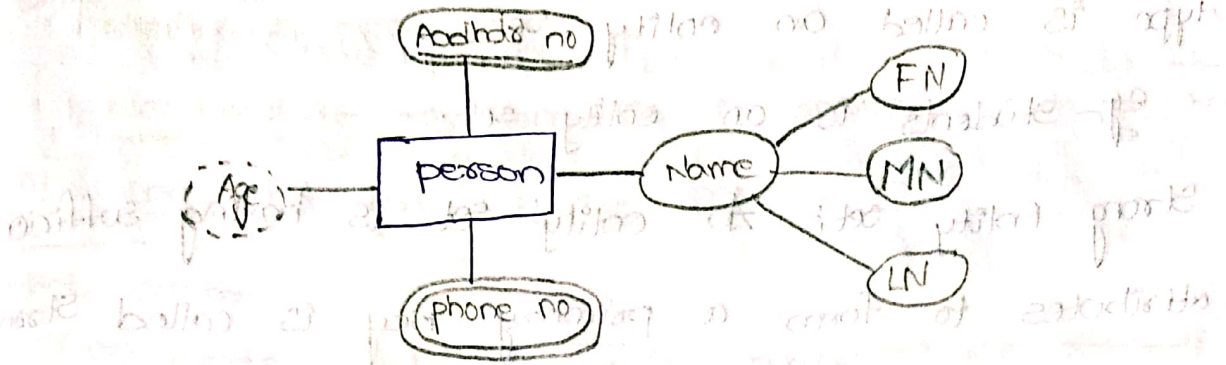


→ specialization

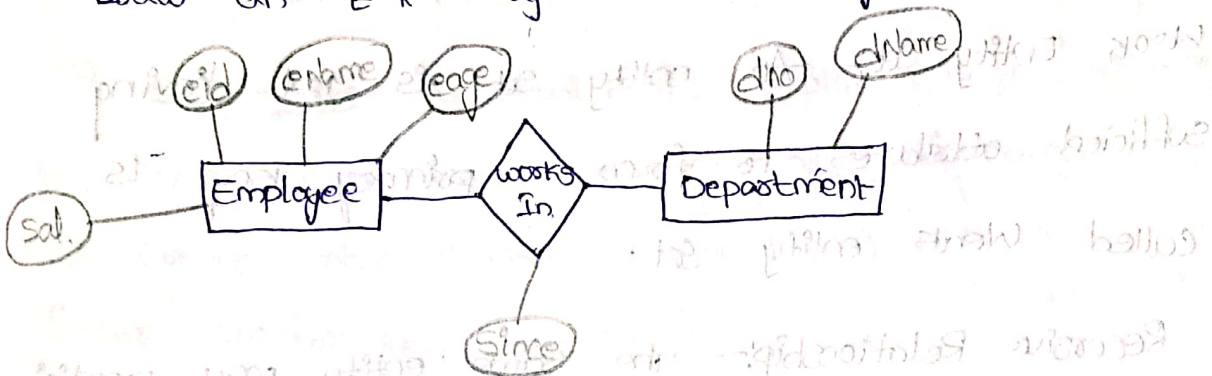
12/1/18

Ex 6

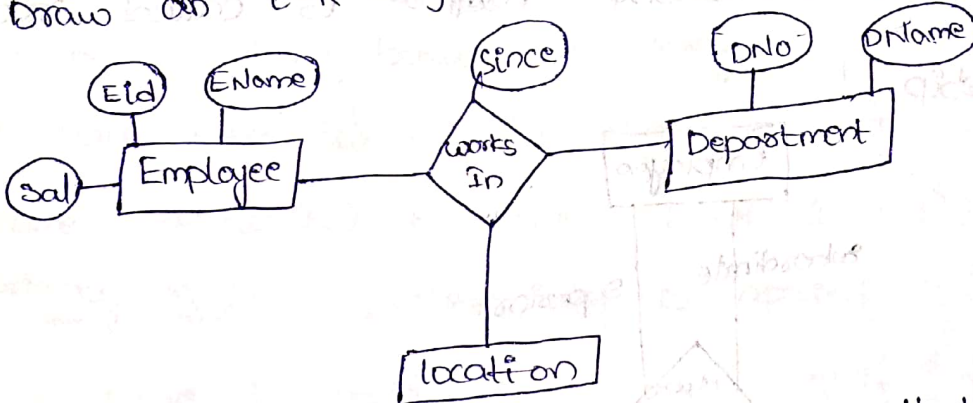
Draw an E-R diagram for person entity?
person (Address No, Name, phone No, Age)



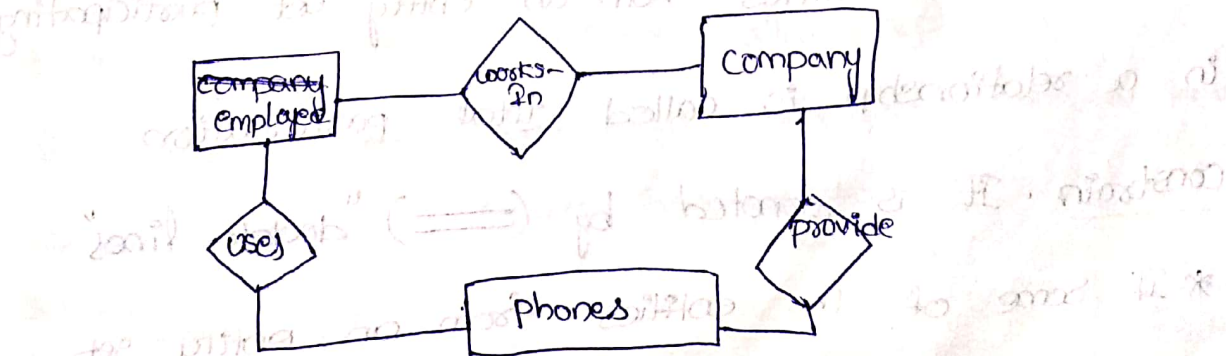
Draw an E-R diagram for Binary Relationship



Draw an E-R diagram for Ternary Relationship



Draw an E-R diagram for the data-base that keeps tracks of company employee phones.



14/7/18

Entity Set:- collection of entities of the same type is called an entity set.

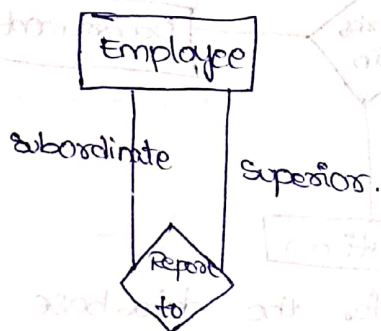
eg:- students is an entity set.

Strong Entity Set:- An entity set is having sufficient attributes to form a primary key is called Strong entity set.

Weak Entity Set:- An entity set is not having sufficient attributes to form a primary key is called Weak entity set.

Recursive Relationship:- The same entity may participate in a relationship itself & is called recursive relationship.

eg:-

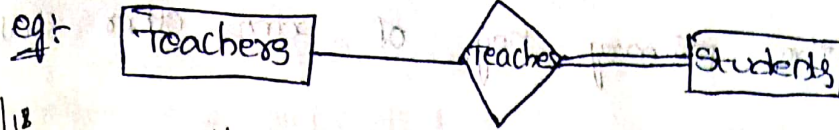


Note:-

* If all the entities from an entity set participating in a relationship is called total participation

constraint. It is denoted by $(=)$ "double lines"

* If some of the entities from an entity set participating in a relationship is called partial participation constraint. It is denoted by "single line" $(-)$



16/7/18

Unit-3

Integrity constraints:-

1) NOT NULL:- This constrain is applied to a column, it means that you can not pass a null value to that column

ex:- Create table Student (SID number(5) NOT NULL, Sname varchar(20) NOT NULL);

2) Unique:- This constrain is applied to a column, it means that column will not allow duplicates.

ex:- Create table Student (SID number(5) NOT NULL UNIQUE, Sname varchar(20) NOT NULL);

3) CHECK:- This constrain is used to restrict the value of a column between a range

ex:- Create table Student (SID number(5) NOT NULL UNIQUE, Sname varchar(20) NOT NULL CHECK (SID > 3));

4) Primary KEY:- A primary key is applied to a column, in a table, is used to uniquely identify each row in a table.

* A table can have only one primary key.

* A primary key will not allow duplicates.

ex:- Create table Student (SID number(5) PRIMARY KEY, Sname varchar(20), Branch varchar(20));

5) FOREIGN KEY:-

* FOREIGN KEY represents relation ship between the tables

* A FOREIGN KEY of a column, whose values are

derived from the primary key of some other table.

Syntax:-

Create table table_name (col1 datatype (size) PRIMARY KEY,
col2 datatype (size), col3 datatype (size) FOREIGN KEY
REFERENCES Another_Table_Name);

17/7/18

KEY CONSTRAINTS:-

Keys:- A key allows us to identify a set of attributes that are enough to differentiate entity.

Super key:- A super key is a set of one (or) more attributes that taken collectively, allows us to

identify uniquely an entity in the entity set

ex:- SID (Student entity set)

candidate key:- A super key which has no proper subset is called a candidate key.

ex:- SID

primary key:- It is the candidate key is used for identifying and accessing the records.

Composite key:- It is a key which requires more than one attribute

Alternate key:- It is a candidate key not be used for primary key.

Secondary key:- The set of attributes commonly used for accessing records but not necessarily

unique.

Foreign Key Constraint:-

* A Foreign Key is a key used to link two tables together.

* A Foreign Key is a field in one table that refers to the primary key in another table

* The Table containing the foreign key is called child table and the table containing the candidate key is called parent table.

ex:- persons(PID, names, Age)

Order (OID, Oname, PID)

From the above tables.

* PID is primary key in persons table

* OID is primary key in orders table.

* PID is foreign key in orders table

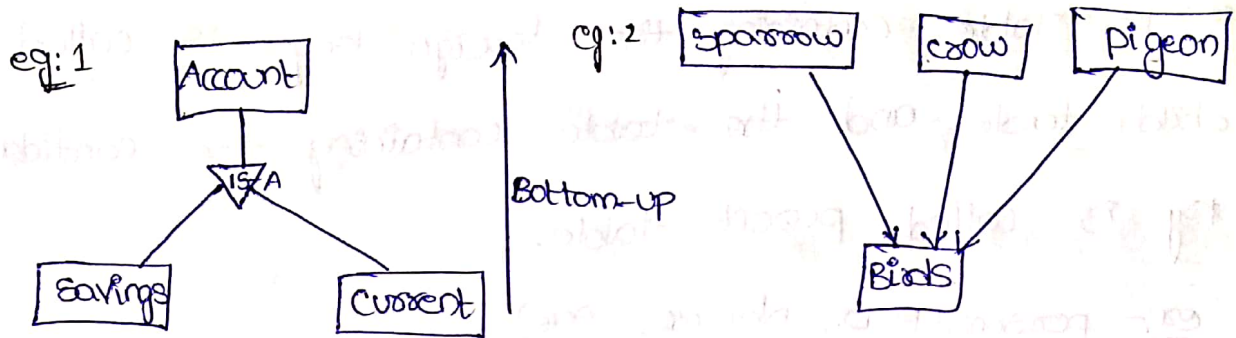
Syntax:-

1) create table persons(PID number(5) PRIMARY NOT NULL
PRIMARY KEY, Name varchar(20), Age number(3));

2) create table orders(OID number(5) NOT NULL PRIMARY
KEY, Oname varchar(20), PID number(5) Foreign Key
References person(PID));

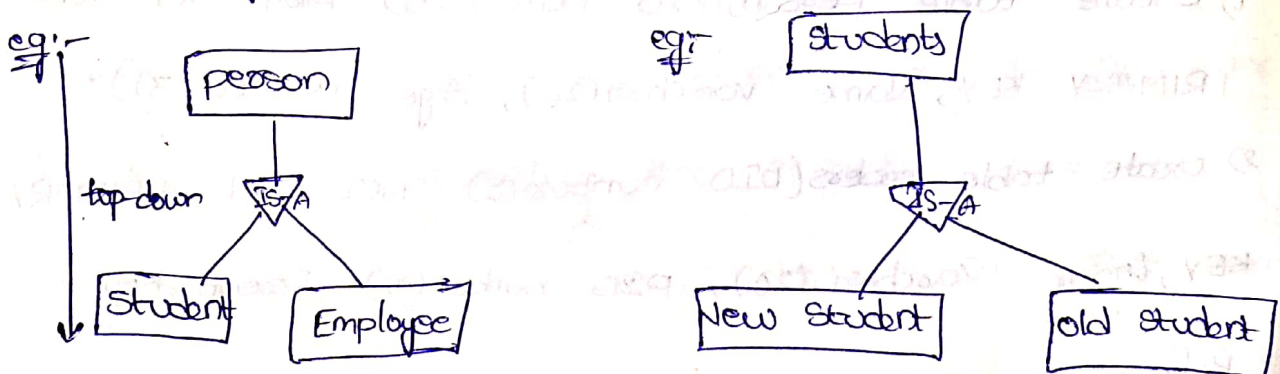
9/1/19
Generalization:- It is Bottom up approach, in which two lower level entities combine to form a higher level entity.

* In generalization a no of entities are brought together into one generalized entity based on their similar characteristics.



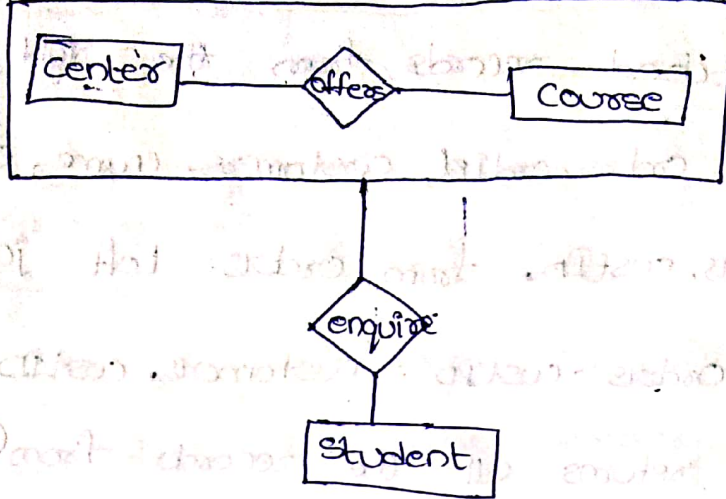
Specialization:- It is a Top down approach, in which one higher level entity can be broken down into two lower level entities.

* In specialization a group of entities is divided into sub-groups based on their characteristics.



Aggregation:-

When relation between two entities is treated as a single entity called as aggregation.



21/7/18

6M

Joins in SQL :-

* SQL joins are used to combine records from two (or) more tables based on a common field b/w them.

* The most common type of join is simple/Equi/

Inner join

customers.

Orders

ordID	cusID	ordname
111	10	Watch
152	2	Mobile
100	5	Dress
175	7	Laptop
222	18	Books
130	29	Rings

cusID	CName	Address
5	cherchu	angole
7	vyslu	Nellore
9	Bhavani	Chennai
10	Sai	Hyderabad
21	Sridhar	Nijaywada
15	Manju	Kandukur

Types of joins:-

1) Inner join ^{simple/Equi}:- It returns all the rows when there is atleast one match in both tables.

ex: SQL > Select orders. ordID, customers. Cname, orders. Ordname
customers. cusID from orders Inner join customers

ON orders. cusID = customers. cusID;

ii) Left join:- It returns all the records from left table and matched records from the right table.

eg:- SQL > select Orders.OrdId, Customers.cname, Orders.
OrdName, Customers.custID from Orders Left join
Customers ON Orders.custID = Customers.custID;

iii) Right join:- It returns all the records from right table and matched records from left table.

eg:- SQL > select Orders.OrdId, Customers.cname, Orders.
OrdName, Customers.custID from Orders Right join
Customers ON Orders.custID = Customers.custID;

iv) Full join:- It returns all the records from left and right tables.

eg:- SQL > select Orders.OrdId, Customers.cname, Orders.
OrdName, Customers.custID from Orders Full join
Customers ON Orders.custID = Customers.custID;

v) Cross join:- It combines all the records from the first table with every record from the right table.

eg:- SQL > select Emp.^{eid}EmpId, Emp.Ename, Dept.Deptno,
Dept.Dname from Emp Cross join Dept ON
Emp.Deptno = Dept.Deptno ;

Self joins: *Joining a table itself is called self join.

join

*This is a type of "Inner join", where both columns belongs to the same table.

*In this join we need to open two copies of a same table in same memory.

*Since the table name is same for both instances, we use table Aliases to make identical copies of same table, to open in different memory

locations.

eg:- SQL > Select e1.EmpId, e1.ename as Name,
e2.ename as Lastname from Employee E1 Inner Join
Employee E2 ON employees.EmpId = E2.EmpId;

23/7/18

Like predicate:-

*This allows comparison of one string value with another string value which is not identical.

*This is achieved by using wild card characters (% , _)

*Two types of wild card characters are available.

i) % :- It allows to match any string of any length.

ii) _ :- It allows to match only one character.

Sid	Sname
S1	Sai
S2	Sridhar
D1	cherchu

In predicate:- In any case a value needs to be compared with a list of values, then "in predicate" is used

eg: select * from Student where Sname In ('sai', 'Sai', 'chenchu', 'vyshu');

Arithmetic Operators:-

* Oracle allows arithmetic operators, to be used while viewing the records from a table (or) while performing manipulation operations such as insert, update, delete.

* The following are the arithmetic operators.

Addition (+), subtraction (-), multiplication (*), division (/)

SNO	SName	Marks
561	Sai	500
562	Lakshmi	550
563	Vyshu	600
564	Bhanu	580
565	Sridhar	520

eg:- List out the student details, after adding marks 60 to each and every student.

→ select * from Student where Marks = Marks + 60;

→ select * from Student where marks = marks - 60;

→ select * from student where marks = marks * 2;

→ select * from Student where marks = marks / 2;

25/7/18 Logical Operators:-

The following are the logical operators used in Oracle

1) AND operator:- This operator allows to create an SQL statement based on two (or) more conditions being met

2) OR operator:- This operator allows to create an SQL statement, where records are written when any one of the condition being met.

3) NOT operator:- This operator display the records from a table that do not satisfy the condition

4) BETWEEN operator:- This operator is used to select the data within a range of values Student

Eg:1 list out the student details with marks and age,

SID	SName	marks	Age
61	Ram	400	20
62	Raju	450	24
63	Ravi	500	22
64	Rakesh	550	21

where the marks between a range of 410 to 510 & age b/w a range of 21 to 24.

Ans:- select marks, age from student where (marks ≥ 410 AND marks ≤ 510) AND (Age ≥ 21 AND Age ≤ 24);

Eg:2 List out the student names where the names are Ram or Rakesh or Chenchu or Sai.
select Sname from Student where Sname = ('Ram' OR 'Rakesh' OR 'Chenchu' OR 'Sai');

eg. List out the student details where the age not in a range of 20 or 24.

→ select * from Student WHERE NOT (Age = 20 or Age = 24)

eg. List out the student details with name, age where the age is in the range of 23 to 25.

→ select name, age from Student where Age BETWEEN 23 to 25;

UNIT-3

SET operators (OR) Relational SET operators:-

These operators are used to combine the information of similar data type from one (or) more tables.

Data type of corresponding columns in all the select statements should be same.

UNION:- Combining the results of two select statements into one result set and then eliminates only duplicate rows from the result set.

Ex:- select Emp, Ename from Emp where DeptNo = 10

UNION select Emp, Ename from Emp where DeptNo = 20;

UNION ALL:-

Combining the results of two select statements into one result set including duplicates from the result set.

Eg:- select Emp, Ename from Emp where DeptNo = 10 UNION ALL select Emp, Ename from Emp where DeptNo = 20;

INTERSECT:- It returns only those rows that are returned by each of the two select statements.

eg:- select EmpNo, EmpName from Emp where DeptNo=10

INTERSECT select EmpNo, EmpName from Emp where DeptNo=20

MINUS:- It returns the results set of one select

statement and removes those rows that are returned

by second select statement.

eg:- select EmpNo, EmpName from Emp where DeptNo=10 MINUS

select EmpNo, EmpName from Emp where DeptNo=20;

NULL VALUES:-

* "NULL" is a special marker used in SQL to indicate that a data value does not exist in the data base.

* In SQL "NULL" is the term is used to represent a missing value.

* An SQL a "NULL" value in a table is a value in a field that appears to be blank.

* "NULL" value is different than a "zero" value. A field that contains spaces

* The following Syntax is used for "NULL" while creating a table.

Syntax:- create table tableName (col1 Datatype (size)

primary key NOT NULL, col2 Datatype (size) NOT NULL,

col3 Datatype (size));

30/1/18
SUB-QUERIES:- A Sub-Query (or) Inner Query (or)

Nested query is a query with in another SQL query and embedded with in ~~where~~ "WHERE" clause.

* A Sub-Query is used to return data that will be used in the main query as a condition.

* Sub queries can be used with SELECT, INSERT, UPDATE, DELETE statements along with operators. like $=, <, <=, >, >=$ IN, BETWEEN

* The following rules must be used for defining a sub query

* Sub queries must be enclosed with in "parenthesis"

* A sub query can have only one column in the select statement.

* An ORDER BY command can not be used in a sub query

* BETWEEN operators can not be used with a sub query, but can be used with in the sub query.

eg:-

customer

Id	Name	Age	Address	Salary
1.	Chenchu	20	ongole	45,000
2.	Vyshu	19	Nellore	40,000
3.	Bhavana	25	Hyd	50,000
4.	Sridhar	22	Abhim	55,000

Sub Queries with SELECT Statements:-

* The Sub Query can be used with Select Statement, the basic syntax is as follows

Syntax:-

Select columnNames from TableName WHERE colName
OPERATOR (select colName from tableName where
Condition);

Ex:- Select * from customers where ID IN (select ID
from customers where salary > 50,000);

Sub Queries with INSERT Statement :-

* A sub queries can be used with INSERT statement, the INSERT statement used the data returned from the sub query to insert into another table.

ex:- INSERT INTO customerBackup select * from
customers where ID IN (select ID from customers)

Sub Queries with UPDATE statement :-

The sub queries can be used in UPDATE statement, either single (or) multiple columns in a table can be updated.

eg:- UPDATE salary by 0.25 times in customer table

for all the customers whose AGE is ≥ 25

→ UPDATE customer set salary = salary * 0.25 (where

Age IN (select AGE from customer where AGE ≥ 25)

sub Queries with DELETE Statement :-

* The sub queries can also be used with delete statements

eg:- Delete the records from customer table for all the customers whose AGE ≥ 25

→ DELETE from customer where AGE IN (Select AGE from customer where AGE ≥ 25)

Relational Algebra :-

* It is a procedural query language which takes instances as input and yields instances of relation as output

* It uses operators to perform queries.

* An operator can be either unary or binary

* The fundamental operations of relational algebra

are as follows * SELECT (σ)

* PROJECT (π)

* UNION (\cup)

* Set Difference ($-$)

* Cartesian product (\times)

* Rename (ρ)

select Operation :-

It selects tuples ^(rows) that satisfy the given predicate from a relation

Notation :- $\sigma_p(r)$

where 'r' is a relation

'p' is propositional logic formula. which

may use connectors like AND, OR, NOT, these forms
may use relational operators like $=, \neq, >, >=, <, <=$

Ex:- $\sigma_{\text{subject} = \text{'database'}} (\text{Books})$

o/p:- selects tuples from books where subject is database

Project Operation (π):-

It PROJECTS columns (fields) that satisfy a given predicate.

Notation:- $\pi_{A_1, A_2, \dots, A_n} (R)$

* Duplicate rows are automatically eliminated

eg:- $\pi_{\text{subject, author}} (\text{Books})$

UNION operation (\cup):-

* It performs binary union b/w two given relations

and is defined as $R \cup S = \{t / t \in R \text{ or } t \in S\}$

* For a union operation the following conditions must be hold.

1. R and S must have same no. of attributes

2. Attribute domains must be compatible

Ex:- $\pi_{\text{author}} (\text{Books}) \cup \pi_{\text{author}} (\text{Article})$

o/p:- PROJECTS the names of authors who have written a book (or) ^{an} article (or) both

Set Difference ($-$):-

The result set is a tuple which are present in one relation but not in second relation

Notation:- $R - S$

Ex:- $\pi_{\text{author}} (\text{Books}) - \pi_{\text{author}} (\text{Articles})$

BASIC structure of SQL

- * SQL is a structured query language. It is a standard language for communicating with relational database management system. SQL statements are used to retrieve data from the database as well as perform tasks such as adding, updating and deleting data from the database.
- * A typical structure of SQL query is,

```

select A1, A2, A3, ... An
from r1, r2, r3 ... rn
where (p) [predicate]

```

A₁, A₂, ... A_n represents an attributes.

r₁, r₂, ... r_n represents a relation.

p is a predicate (condition).

- * We are using mainly three clauses in a SQL query (select, from, where)

(i) select clause :- It list the attributes desired in the result of query.

- * This clause similar to the projection operation of the relational algebra.

- * SQL names are case-insensitive (we are using upper, lower cases).

- * SQL allows duplicates in relations as well as in query results.

- * Elimination of duplicates, insert the keyword distinct after select clause.

An asterisk(*) in the select clause denotes all attributes.

* select clause used for arithmetic expressions (+, -, *, /) operating on constants (or) attributes of tuples

Ex: tablename: Loan

loan number	Branch name	amount
L-11	Round Hill	9000
L-14	Down town	15000
L-15	Perry ridge	15000
L-16	Down town	9000
L-17	Red word	10,000

1) Query: find the names of all branches in the loan relation.

Query: SQL > select branchname from loan;

o/p: Displays only branchname.

2) find the names of all branches in the loan relation and remove duplicate names.

Query: SQL > select distinct branchname from loan;

o/p:

branchname
Round Hill
Down town
Perry ridge
Red word

3) find the names of all branches in the loan

Query: SQL > select all branch name from loan;

o/p:

Branch name
Round Hill
Down town
Perry ridge
Red word

4) > select * from loan.

o/p: Displays the loan table.

5) List the tuple of loans of relation with amount multiplied by 100.

o/p:

amount
900000
1500000
1500000
900000
1000000

Query: SQL > select amount * 100
from loan;

Where clause:-

- * It specifies condition that the result must satisfy
- * the where clause is similar to the selection operation in relational algebra.
- * It is the expression that controls the which rows appear in the resulting table.
- * the SQL uses (AND, OR) operators, Comparison operators (<, >, <=, >=, ==) and also used b/w, b/w and, not b/w operators.

Ex:- Find the loan number from the relation loan with loan amount b/w 10,000 to 15000.

SQL > select loan number from loan where amount b/w 10,000 and 15000;

o/p:

Loan number
L-14
L-15
L-16
L-17

From clause:- It specifies the table names that will be queried to retrieve the desired data.

Ex:- Find the all attributes from the loan relation.

Query: > select * from loan;
Display the loan table.

and set operations in SQL:-
 set operators are used to compile the results of two (or) more select statements.
 SQL supports few set operations which can be performed on the table data.
 * These are used to get meaningful results from data stored in the table.

the set operations are,
 union, unionall, Intersect, minus (set difference),
 except, exceptall

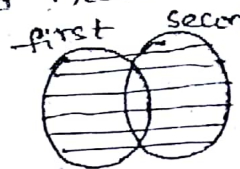
union:- It is used to combine the result of two (or) more select statements. It will eliminate duplicate rows from its result set. If you want perform union operation, we must follow the rules.

- * same no. of columns should be selected from both the tables.
- * the same no. of column expressions.
- * the same data type, and have them in the same order.

Ex:-

ID	Name
1	abhi
2	adam

ID	Name
1	adam
2	chester



query:- select * from first;
 union
 select * from second;

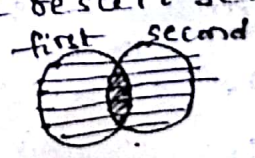
ID	Name
1	abhi
2	adam
3	chester

union all:- This operation is similar to union. But, it allows duplicate rows to the result set.

Ex:-

ID	Name
1	abhi
2	adam

ID	Name
1	adam
2	chester

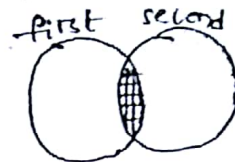


Query ÷ > select * from first;
union all
> select * from second;

ID	Name
1	Abhi
2	Adam
2	Adam
3	Chester

Intersect ÷ It is used to combine two select statements but it only returns the records which are common from both select statement (In case of Intersect the no. of columns and data type must be same in both tables)

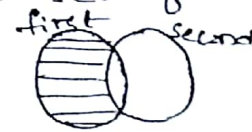
Query ÷ > select * from first;
Intersect
> select * from second;



ID	Name
2	Adam

Minus ÷ The minus operation combines the results of two select statements and return only those in the final result which belongs to the first set of the results (first table data).

Query ÷ > select * from first;
minus
> select * from second;



ID	Name
1	Abhi

Except ÷ It was introduced in 2005. It is used to achieve 'distinct' and 'not in' queries. Except operator returns all distinct rows from Left hand side table which doesn't exist in right hand side table.

Query ÷ > select * from first;
Except
> select * from second;



ID	Name
1	Abhi

Except all ÷ Which returns all records from the first table which are not written in second table. but except all operator doesn't remove duplicate records.

query: \rightarrow select * from first;

except all

\rightarrow select * from second;

first second



Id	name
1	abhi
1	abhi

- Nested / Sub / Inner query :- A sub query (or inner query) (or) a nested query is a query within another SQL query and embedded with in the where clause. The inner query is used to determine the results of the outer query.

Ex: find the name of a student who has minimum age.

query: \rightarrow select name, min(age) from student;

sname	age
A	12
B	19
C	24
D	19
E	30

nested query: \rightarrow select sname, age from student where age \geq (select min(age) from student);

- Aggregate operators :-

Aggregate functions that take a connections of values as input and return a single value to the output. The SQL aggregate operators are.

(1) Avg () :- It returns the Avg value of a numeric column.

syntax: select Avg(columnname) from tablename;

order no	orderid	products	quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

Ex: select Avg(quantity) from order;

(2) min() :- It returns minimum value of a column.

Syntax :- select min(columnname) from tablename;

Ex :- select min(quantity) from order;

(3) max() :- It returns maximum value of a numeric column.

Syntax :- select max(columnname) from tablename;

Ex :- select max(quantity) from order;

(4) sum() :- It returns total sum of a numeric column.

Syntax :- select sum(columnname) from tablename;

Ex :- select sum(quantity) from order;

(5) count() :- It returns the no. of rows that matches a specified criteria.

Syntax :- select count(columnname) from tablename;

Ex :- select count(quantity) from order;

Correlated Query (or) Subquery :-

The subquery is a query that contains inside another query. The inner query is used to determine the results of outer query. In this correlated nested queries, the outer query is executed first.

Difference b/w nested query, correlated nested query :-

Nested query

* Inner query is executed first.

* Inner query is executed only once and result is used by outer query.

Correlated nested query

* Outer query is executed first.

* Inner query is executed for each of the records that outer query returns.

using comparison operators ($>$, $<$, $>=$, $<=$, $=$), in operator, b/w operator.

* Always used in the where clause.

Ex: \rightarrow select sname, age from student where age = (select min(age) from student);

* uses comparison operators, in, b/w, exists and not exist, any.

* It is used in the where clause as well as columns of select statement with operators.

Ex: SQL \rightarrow select sname, age from student where age exists (select min(age) from student);

• Null values:—

* The SQL NULL is a keyword used to missing value in a field. It also represent blank space (not given any variable).

* A field with a null value is a field with no value.

Ex: Create table custor (ID number not null, name varchar(20) not null, age number, address varchar(20), sal number);

customer				
ID	name	age	address	sal
1	Ramesh	32	Hyd	2000
2	Suresh	25	Delhi	1500
3	Kaushal	23	Mumbai	3000
4	Komal	25	Pune	-
5	Manu	27	Rajasthan	-

Here, not null represents that column should take values, and we do not use not null keyword that column takes null values.

Syntax: SQL \rightarrow select id, name, age, address, sal from customer where sal is not null;

Op:

ID	name	age	address	sal
1	Ramesh	32	Hyd	2000
2	Suresh	25	Delhi	1500
3	Kaushal	23	Mumbai	3000

Syntax: SQL \rightarrow select id, name, age, address, sal from

o/p:

ID	name	age	address	sal
4	Romal	25	pune	-
5	marce	27	Rajasthan	-

Logical connectivity operators:-

there are three logical operators AND, OR, NOT. This operators compare two conditions at a time to determine whether a row can be selected for the output. when retrieving data using a select statement, we can use the logical operators in the where clause, which allows to combine more than one condition.

AND:-

If we want to select rows in a table that must satisfy all the given condition.

Ex:-> select id, name, age, sal
from customer where
age \geq 25 and age \leq 32;

ID	name	age	address	sal
1	Ramesh	32	Hyd	2000
2	aresh	25	Delhi	1500
3	Koushal	23	mumbai	3000
4	Romal	25	pune	4000
5	marce	27	Rajasthan	4500

OR:-

If you want to select rows that satisfying atleast one of the given conditions.

Ex:-> select id, name, age, sal
from customer where name = 'Ramesh'
or name = 'Koushal';

ID	name	age	sal
1	Ramesh	32	2000
3	Koushal	25	3000

NOT:-

If you want to find rows that don't satisfy a condition we can use logical operator NOT. The NOT operator rejects in the result of a condition. That

if a condition is satisfy then the row is not returned.

Ex: select id, name, age, sal from customers where not sal = 7000;

Complex integrity constraints:

constraints over single table: - complex constraints over a single table (or) specified using a table constraints. which has the form check conditional expression.

syntax: - create table sailors (sid int, sname varchar(10), rating int, age int, primary key (sid), check (rating >= 1 and rating <= 10));

Domain Constraints:

A user can define a new domain using the create domain statement, which makes of check constraints.

syntax: - create domain age int default 18;

syntax: - create domain age int check (age >= 18 and age <= 25);

Integer is the datatype of a age domain that can be restricted by check constrain. In this above example age attribute is the domain.

Assertion: (ICS over several table)

When a constrain involves two (or) more tables, the mechanism becomes so difficult to overcome this problem the SQL introduced the concept assertion. Assertions are constraints which are not associated with any one table.

Syntax :-

SQL > create table sailors (sid int, sname var^{character}
rating int, age int, primarykey(sid), ⁽¹⁰⁾ check
(rating >= 1 and rating <= 10), check((select
count (s.sid from sailors s) + (select count
(p.pid) from person) < 100));

using Assertion :-

Syntax :- create assertion ^{assertion name} smallclub

(10),
check ((select count (s.sid) from student s) +
(select count (p.pid) from person p) < 100);

Triggers in active data base :-

- * Triggers are stored programs, which are automatically executed (or) fired when some events occur.
- * A trigger can be defined as a program that is executed by DBMS whenever updations ~~are~~ specified on database tables.
- * It is like an event which occurs whenever a change is done to the tables (or) columns of the tables.
- * only DBA can specifies the triggers.
- * Triggers are written to be executed in response to any of the following events.
 - (i) A database manipulation ^(DML) statement (Delete, insert, update).
 - (ii) Data definition statement (DDL) -> (Create, Alter, Drop)
 - (iii) A Database operations -> server error, log on, log off, startup, shutdown
- * Trigger could be define on the table, view, schema (or) database with which the event is associated.

General form of the trigger:-

the general form of the trigger includes the following.

i) Event

ii) Condition

iii) Action

Event:- It describes the modifications done on the database which lead to the activation of trigger. The events are DDL, DML, Database operations.

Condition:- These are used to specify the actions to be taken when the corresponding event occurs and the condition evaluates true then respect to action to be taken otherwise, action is rejected.

Action:- Action specifies the action to be taken place at when the corresponding event occurs and the condition evaluates to true. An action is a collection of SQL statements that are executed as a part of trigger activation.

Structure of Trigger:-

create or replace trigger <triggername>
Before or after or Instead of

Insert or update or Delete on <tablename>
for each row

which <condition for trigger to get execute>
declare

<declaration part>

begin

<execution part>

-Exception

<exception part>

end;

The following program creates a row level trigger for the customer table that would be fired for insert (or) update (or) delete operations performed on customer table. This trigger will display the salary difference b/w old and new salary statements.

create or replace trigger SalDiff

Before
delete or insert or update on customer
for each row
when (new.ID > 0)

customer			
ID	name	age	Sal
1	xyz	23	2000
2	PS9	25	5000
3	abc	32	3000

Declare

sal_difference number;

begin

sal_difference := :new.sal - :old.sal;

dbms_output.put_line ('oldsal:', ||:old.sal);

dbms_output.put_line ('newsal:', ||:new.sal);

dbms_output.put_line ('saldifference:', ||:sal_diff);

end;

/

→ Trigger Created.

→ We want to insert a row in a customer table.

Insert Query:

Insert into customer values ('4', 'neha', '33', 5000);

oldsal := —

newsal := 5000

saldiff := —

update the table, the salary incremented by 500

Types of Triggers :-

Triggers are classified based on levels they are two levels of Triggers to be invoked. They are.

statement-level trigger:- These triggers executed only once for multiple rows which are effected by trigger action.

row-level trigger:- These triggers executed by for each row of the table that is effected by the event of triggers. The row level triggers are.

i) Before trigger

ii) After trigger

Before trigger:- It execute before the triggering DML statements.

After trigger:- It execute After the triggering DML statements executed.

Benefits of triggers :-

- * Generating some derived column values automatically.
- * Enforcing referential integrity.
- * Event logging and storing information and table Access.
- * Auditing
- * synchronous replication of tables.
- * Imposing security authorization.
- * preventing invalid transactions.

UNIT-4

SCHEMA REFINEMENT and NORMAL FORMS

Syllabus: Problems caused by redundancy, Decompositions, problem related to decomposition, reasoning about FDS, FIRST, SECOND, THIRD Normal forms, BCNKF, Lossless join Decomposition, Dependency preserving Decomposition, Schema refinement in Data base Design, Multi valued Dependencies, FORTH Normal Form.

Schema Refinement: The purpose of Schema refinement is used for a refinement approach based on decompositions. Redundant storage of information (i.e. duplication of data) is main cause of problem. This redundancy is eliminated by decompose the relation.

Q. Problems caused by redundancy: Redundancy is a method of storing the same information repeatedly. It means, storing the same data more than one place with in a database is can lead several problems. Such as

1) **Redundant Storage:** It removes the multi-valued attribute. It means, some tuples or information is stored repeatedly.

2) **Update Anomalies:** Suppose, if we update one row (or record) then DBMS will update more than one similar row, causes update anomaly.

For example, if we update the department name those who getting the salary 40000 then that will update more than one row of employee table is causes update anomaly.

3) **Insertion Anomalies:** In insertion anomaly, when allows insertion for already existed record again, causes insertion anomaly.

4) **Deletion Anomalies:** In deletion anomaly, when more than one record is deleted instead of specified or one, causes deletion anomaly.

Consider the following example,

Eno	ename	salary	rating	hourly_wages	hours_worked
111	suresh	25000	8	10	40
222	eswar	30000	8	10	30
333	sankar	32000	5	7	30
444	padma	31000	5	7	32
555	aswani	35000	8	10	40

In this example,

1) **Redundancy storage:** The rating value 8 corresponds to the hourly_wages 10 and there is repeated three times.

2) **Update anomalies:** If the hourly_wages in the first tuple is updated, it does not make changes in the corresponding second and last tuples. Because, the key element of tuples is emp_id.

i.e. update employee set hourly_wages = 12 where emp_id = 140; But the question is update hourly_wages from 10 to 12.

3) **Insertion anomalies:** If we have to inset a new tuple for an employee, we should know both the values rating value as well as hourly_wages value.

4) **Deletion anomalies:** Delete all the tupes through a given rating value, causes deletion anomaly.

Q. Decompositions: A relation schema (table) R can be decomposed into a collection of smaller relation schemas (tables) (i.e. R_1, R_2, \dots, R_m) to eliminate anomalies caused by the redundancy in the original relation R is called Decomposition of relation. This shown in Relational Algebra as

$$R_i \subseteq R \text{ for } 1 \leq i \leq m \text{ and } R_1 \cup R_2 \cup R_3 \dots R_m = R.$$

(or)

A decomposition of a relation schema R consists of replacing the relation schema by two or more relation schemas that each contains a subset of the attributes of R and together include all attributes in R.

(or)

In simple, words, "The process of breaking larger relations into smaller relations is known as decomposition".

Consider the above example that can be decomposing the following hourly_emps relation into two relations.

Hourly_emp (eno, ename, salary, rating, hourly_wages, hours_worked)

This is decomposed into

Hourly_empsd(eno, ename, salary, rating, hours_worked) and

Wages(rating, hourly_wages)

Eno	ename	salary	rating	hours_worked
111	suresh	25000	8	40
222	eswar	30000	8	30
333	sankar	32000	5	30
444	padma	31000	5	32
555	aswani	35000	8	40

rating	hourly_wages
8	10
5	7

Q. Problems Related to Decomposition:

The use of Decomposition is to split the relation into smaller parts to eliminate the anomalies. The question is

1. What are problems that can be caused by using decomposition?
2. When do we have to decompose a relation?

The answer for the **first** question is, two properties of decomposition are considered.

- 1) **Lossless-join Property:** This property helps to identify any instance of the original relation from the corresponding instance of the smaller relation attained after decomposition.
- 2) **Dependency Preservation:** This property helps to enforce constraint on smaller relation in order to enforce the constraints on the original relation.

The answer for the **second** question is, number of normal forms exists. Every relation schema is in one of these normal forms and these normal forms can help to decide whether to decompose the relation schema further or not.

The **Disadvantage** of decomposition is that it enforces us to join the decomposed relations in order to solve the queries of the original relation.

Q. What is Relation?: A relation is a named two-dimensional table of data. Each relation consists of a set of named columns and an arbitrary number of unnamed rows.

For example, a relation named Employee contains following attributes, emp-id, ename, dept name and salary.

Emp-id	ename	dept name	salary
100	Simpson	Marketing	48000
140	smith	Accounting	52000
110	Lucero	Info-systems	43000
190	Davis	Finance	55000
150	martin	marketing	42000

marketing 42000

What are the Properties of relations?:

The properties of relations are defined on two dimensional tables. They are:

- ❑ Each relation (or table) in a database has a unique name.
- ❑ An entry at the intersection of each row and column is atomic or single. These can be no multiplied attributes in a relation.
- ❑ Each row is unique, no two rows in a relation are identical.
- ❑ Each attribute or column within a table has a unique name.
- ❑ The sequence of columns (left to right) is insignificant the column of a relation can be interchanged without changing the meaning use of the relation.
- ❑ The sequence of rows (top to bottom) is insignificant. As with column, the rows of relation may be interchanged or stored in any sequence.

Q. Removing multi valued attributes from tables: the "second property of relations" in the above is applied to this table. In this property, there is no multi valued attributes in a relation. This rule is applied to the table or relation to eliminate the one or more multi valued attribute. Consider the following the example, the employee table contain 6 records. In this the course title has multi valued values/attributes. The employee 100 taken two courses vc++ and ms-office. The record 150 did not taken any course. So it is null.

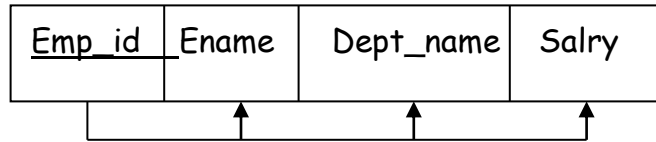
Emp-id	name	dept-name	salary	course_title
100	Krishna	cse	20000	vc++, msoffice
140	Rajasekhar	cse	18000	C++, DBMS, DS

Now, this multi valued attributes are eliminated and shown in the following employee2 table.

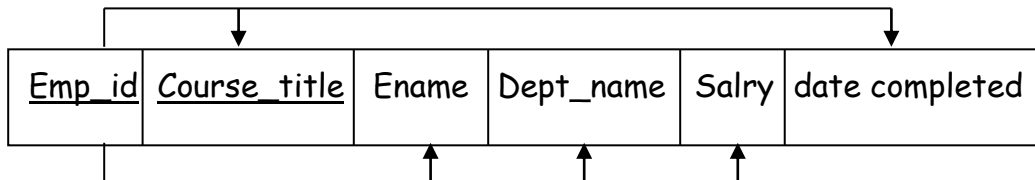
Emp-id	name	dept-name	salary	course_title
100	Krishna	cs	20000	vc++
100	Krishna	cs	20000	MSoffice
140	Rajasekhar	cs	18000	C++
140	Rajasekhar	cs	18000	DBMS
140	Rajasekhar	cs	18000	DS

Functional dependencies : A functional dependency is a constraint between two attributes (or) two sets of attributes.

For example, the table EMPLOYEE has 4 columns that are Functionally dependencies on EMP_ID.



Partial functional dependency : It is a functional dependency in which one or more non-key attributes are functionally dependent on part of the primary key. Consider the following graphical representation, in that some of the attributes are partially depend on primary key.



In this example, Ename, Dept_name, and salary are fully functionally depend on Primary key of Emp_id. But Course_title and date_completed are partial functional dependency. In this case, the partial functional dependency creates redundancy in that relation.

Q. What is Normal Form? What are steps in Normal Form?

NORMALIZATION: Normalization is the process of decomposing relations to produce smaller, well-structured relation.

To produce smaller and well structured relations, the user needs to follow six normal forms.

Steps in Normalization:

A normal form is state of relation that result from applying simple rules from regarding functional dependencies (relationships between attributes to that relation. The normal form are

1. First normal form
2. Second normal form
3. Third normal form
4. Boyce/codd normal form
5. Fourth normal form
6. Fifth normal form

- 1) **First Normal Form**: Any multi-valued attributes (also called repeating groups) have been removed,
- 2) **Second Normal Form**: Any partial functional dependencies have been removed.
- 3) **Third Normal Form**: Any transitive dependencies have been removed.
- 4) **Boyce/Codd Normal Form**: Any remaining anomalies that result from functional dependencies have been removed.
- 5) **Fourth Normal Form**: Any multi-valued dependencies have been removed.
- 6) **Fifth Normal Form**: Any remaining anomalies have been removed.

Advantages of Normalized Relations Over the Un-normalized Relations: The advantages of normalized relations over un-normalized relations are,

- 1) Normalized relation (table) does not contain repeating groups whereas, un-normalized relation (table) contains one or more repeating groups.
- 2) Normalized relation consists of a primary key. There is no primary key presents in un-normalized relation.
- 3) Normalization removes the repeating group which occurs many times in a table.
- 4) With the help of normalization process, we can transform un-normalized table to First Normal Form (1NF) by removing repeating groups from un-normalized tables.
- 5) Normalized relations (tables) gives the more simplified result whereas un-normalized relation gives more complicated results.
- 6) Normalized relations improve storage efficiency, data integrity and scalability. But un-normalized relations cannot improvise the storage efficiency and data integrity.
- 7) Normalization results in database consistency, flexible data accesses.

Q. FIRST NORMAL FORM (1NF): A relation is in first normal form (1NF) contains no multi-Valued attributes. Consider the example employee, that contain multi valued attributes that are removing and converting into single valued attributes

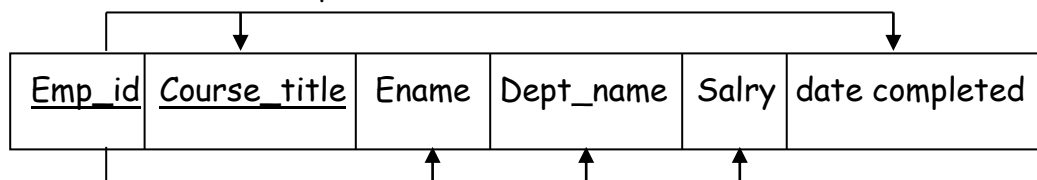
Multi valued attributes in course title

Emp-id	name	dept-name	salary	course_title
100	Krishna	cse	20000	vc++, msoffice
140	Raja	it	18000	C++, DBMS, DS

Removing the multi valued attributes and converting single valued using First NF

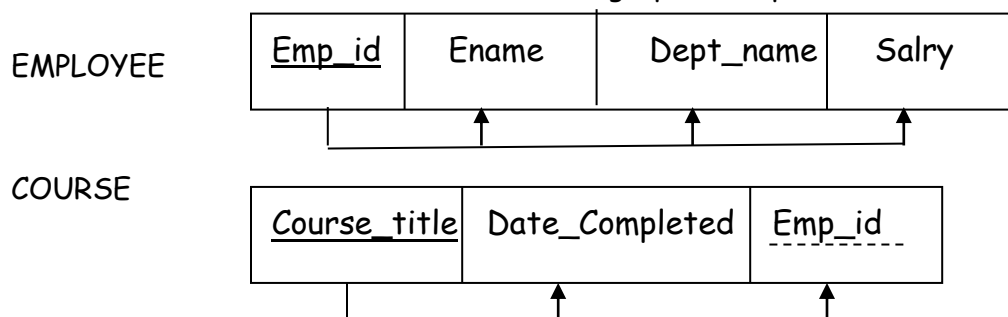
Emp-id	name	dept-name	salary	course_title
100	Krishna	cse	20000	vc++
100	Krishna	cse	20000	msoffice
140	Raja	it	18000	C++
140	Raja	it	18000	DBMS
140	Raja	it	18000	DS

SECOND NORMAL FORM(2NF): A relation in Second Normal Form (2NF) if it is in the 1NF and if all non-key attributes are fully functionally dependent on the primary key. In a functional dependency $X \rightarrow Y$, the attribute on left hand side (i.e. x) is the primary key of the relation and right side attributes on right hand side i.e. Y is the non-key attributes. In some situation some non key attributes are partial functional dependency on primary key. Consider the following example for partial functional specification and also that convert into 2 NF to decompose that into two relations.



In this example, Ename, Dept_name, and salary are fully functionally depend on Primary key of Emp_id. But Course_title and date_completed are partial functional dependency. In this case, the partial functional dependency creates redundancy in that relation.

- To avoid this, convert this into Second Normal Form. The 2NF will decompose the relation into two relations, shown in graphical representation.

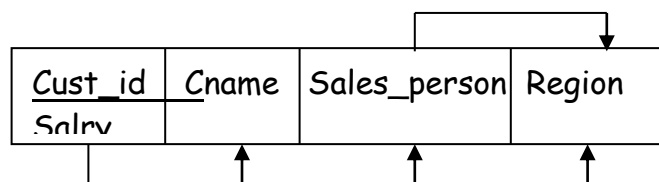


In the above graphical representation

- the EMPLOYEE relation satisfies rule of 1 NF in Second Normal form and
- the COURSE relation satisfies rule of 2 NF by decomposing into two relation.

THIRD NORMAL FORM(3NF): A relation that is in Second Normal form and has no transitive dependencies present.

Transitive dependency : A transitive is a functional dependency between two non key attributes. For example, consider the relation Sales with attributes cust_id, name, sales person and region that shown in graphical representation.



<u>CUST ID</u>	<u>NAME</u>	<u>SALESPERSON</u>	<u>REGION</u>
1001	Anand	Smith	South
1002	Sunil	kiran	West
1003	Govind	babu rao	East
1004	Manohar	Smith	South
1005	Madhu	Somu	North

In this example, to insert, delete and update any row that facing Anomaly.

- Insertion Anomaly:** A new salesperson is assigned to North Region without assign a customer to that salesperson. This causes insertion Anomaly.
- Deletion Anomaly:** If a customer number say 1003 is deleted from the table, we lose the information of salesperson who is assigned to that customer. This causes, Deletion Anomaly.
- Modification Anomaly:** If salesperson Smith is reassigned to the East region, several rows must be changed to reflect that fact. This causes, update anomaly.

To avoid this Anomaly problem, the transitive dependency can be removed by decomposition of SALES into two relations in **3NF**.

Consider the following example, that removes Anomaly by decomposing into two relations.

CUST_ID	NAME	SALESPERSON	SALESPERSON	REGION
1001	Anand	Smith	Smith	South
1002	Sunil	kiran	kiran	West
1003	Govind	babu rao	babu rao	East
1004	Manohar	Smith	Smith	South
1005	Madhu	Somu	Somu	North

SALES PERSON

Sales_person	Region
--------------	--------

CUSTOMER

<u>Cust_id</u>	Cname	Salry
----------------	-------	-------

Q. BOYCE/CODD NORMAL FORM(BCNF): A relation is in BCNF if it is in 3NF and every determinant is a candidate key.

FD in F^+ of the form $X \rightarrow A$ where $X \subseteq S$ and $A \in S$, X is a super key of R .

Boyce-Codd normal form removes the remaining anomalies in 3NF that are resulting from functional dependency, we can get the result of relation in BCNF.

For example, STUDENT-ADVISOR IN 3NF

STUDENT-ADVISOR

Student-id	major-subject	faculty-advisor
------------	---------------	-----------------

STUDENT-ADVISOR relation with simple data.

<u>STUDENT-ID</u>	<u>MAJOR-SUBJECT</u>	FACULTY-ADVISOR
1	MATHS	B
2	MATHS	B
3	MATHS	B
4	STATISTICS	A
5	STATISTICS	A

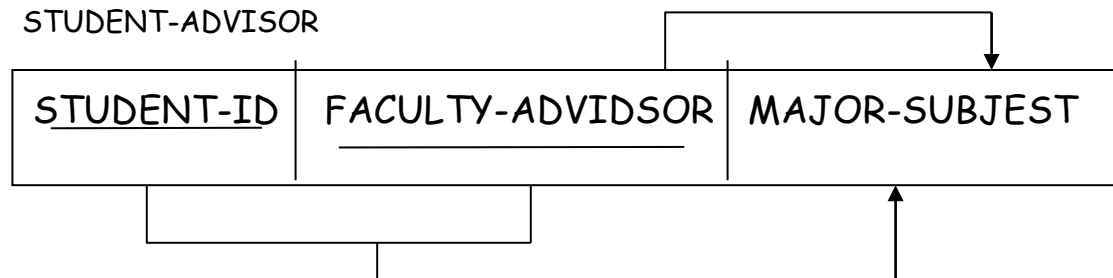
In the above relation the primary key is student-id and major-subject. Here the part of the primary key major-subject is dependent upon a non key attribute faculty-advisor. So, here the determinant is the faculty-advisor. But it is not a candidate key.

Here in this example there are no partial dependencies and transitive dependencies. There is only functional dependency between part of the primary key and non key attribute. Because of this dependency there is an anomaly in this relation.

Suppose that in maths subject the advisor 'B' is replaced by X. This change must be made in two or more rows in this relation. This is an update anomaly.

To convert a relation to BCNF the first step in the original relation is modified that the determinant(non key attributes) becomes a component of the primary key of new relation. The attribute that is dependent on determinant becomes a non key attributes .

STUDENT-ADVISOR



The second step in the conversion process is decompose the relation to eliminate the partial functional dependency. This results in two relations. these relations are in 3NF and BCNF . since there is only one candidate key. That is determinant.

Two relations are in BCNF.

ADVISOR

<u>Faculty- advisor</u>	major-subject
-------------------------	---------------

STUDENT

<u>Student-id</u>	faculty-advisor
-------------------	-----------------

In these two relations the student relation has a composite key , which contains attributes student-id and faculty-advisor. Here faculty-advisor a foreign key which is referenced to the primary key of the advisor relation.

Two relations are in BCNF with simple data.

<u>Faculty Advisor</u>	Major_subject_
B	MATHS
A	PHYSICS

Student_id	Faculty_Advisor
1	B
2	B
3	A
4	A
5	A

Q.Fourth Normal Form (4 NF) : A relation is in BCNF that contain no multi-valued dependency. In this case, 1 NF will repeated in this step. For example, R be a relation schema, X and Y be attributes of R, and F be a set of dependencies that includes both FDs and MVDs. (i.e. Functional Dependency and Multi-valued Dependencies). Then R is said to be in Fourth Normal Form (4NF) if for every MVD $X \twoheadrightarrow Y$ that holds over R, one of the following statements is true.

1) $Y \subseteq X$ or $XY = R$, or 2) X is a super key.

Example: Consider a relation schema ABCD and suppose that are FD $A \rightarrow BCD$ and the MVD $B \twoheadrightarrow C$ are given as shown in Table

It shows three tuples from relation ABCD that satisfies the given MVD $B \twoheadrightarrow C$. From the definition of a MVD, given tuples t_1 and t_2 , it follows that tuples t_3 must also be included in the above relation. Now, consider tuples t_2 and t_3 . From the given FD $A \rightarrow BCD$ and the fact that these tuples have the same A-value, we can compute

B	C	A	D	tuples
b	c ₁	a ₁	d ₁	- tuple t ₁
b	c ₂	a ₂	d ₂	- tuple t ₂
b	c ₁	a ₂	d ₂	- tuple t ₃

the $C_1 = C_2$. Therefore, we see that the FD $B \rightarrow C$ must hold over ABCD whenever the FD $A \rightarrow BCD$ and the MVD $B \twoheadrightarrow C$ holds. If $B \rightarrow C$ holds, the relation is not in BCNF but the relation is in 4 NF.

The fourth normal form is useful because it overcomes the problems of the various approaches in which it represents the multi-valued attributes in a single relation.

Q. Fifth Normal Form (5 NF) : Any remaining anomalies from 4 NF relation have been removed.

A relation schema R is said to be in Fifth Normal Form (5NF) if, for every join dependency $* (R_1, \dots, R_n)$ that holds over R, one of the following statements is true.

* $R_i = R$ for some i , or

* The JD is implied by the set of those FDs over R in which the left side is a key for R. It deals with a property loss less joins.

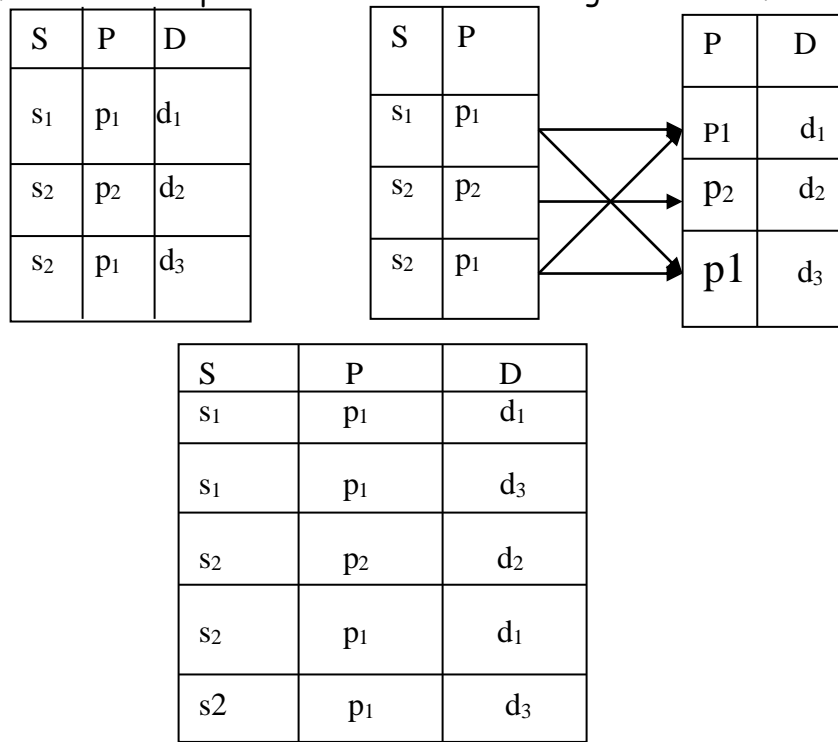
Q. LOSSELESS-JOIN DECOMPOSITION:

Let R be a relation schema and let F be a set FDs (Functional Dependencies) over R. A decomposition of R into two schemas with attribute X and Y is said to be lossless-join decomposition with respect to F, if for every instance r of R that satisfies the dependencies in F_r .

$$\pi_X(r) \bowtie \pi_Y(r) = r$$

In simple words, we can recover the original relation from the decomposed relations.

In general, if we take projection of a relation and recombine them using natural join, we obtain some additional tuples that were not in the original relation.



The decomposition of relation schema r i.e. SPD into SP i.e. PROJECTING $\pi_{sp}(r)$ and PD i.e., projecting $\pi_{pd}(r)$ is therefore lossless decomposition as it gains back all original tuples of relation ' r ' as well as with some additional tuples that were not in original relation ' r '.

Q. Dependency-Preserving Decomposition:

Dependency-preserving decomposition property allows us to check integrity constraints efficiently. In simple words, a dependency-preserving decomposition allows us to enforce all FDs by examining a single relation on each insertion or modification of a tuple.

Let R be a relation schema that is decomposed in two schemas with attributes X and Y and let F be a set of FDs over R . The projection of F on X is the set of FDs in the closure F^+ that involves only attributes in X . We denote the projection of F on attributes X as F_x . Note that a dependency $U \rightarrow V$ in F^+ is in F_x only if all the attributes in U and V are in X . The decomposition of relation schema R with FDs F into schemas with attributes X and Y is dependency-preserving if $(F_x \cup F_y)^+ = F^+$.

That is, if we take the dependencies in F_x and F_y and compute the closure of their union, then we get back all dependencies in the closure of F . To enforce F_x , we need to examine only relation X (that inserts on that relation) to enforce F_y , we need to examine only relation Y .

Example: Suppose that a relation R with attributes ABC is decomposed into relations with attributes AB and BC . The set F of FDs over r includes $A \rightarrow B$, $B \rightarrow C$ and $C \rightarrow A$. Here, $A \rightarrow B$ is in F_{AB} and $B \rightarrow C$ is in F_{BC} and both are dependency-preserving. Where as $C \rightarrow A$ is not implied by the dependencies of F_{AB} and F_{BC} . Therefore $C \rightarrow A$ is not dependency-preserving.

Consequently, F_{AB} also contains $B \rightarrow A$ as well as $A \rightarrow B$ and F_{BC} contains $C \rightarrow B$ as well as $B \rightarrow C$. Therefore $F_{AB} \cup F_{BC}$ contain $A \rightarrow B$, $B \rightarrow C$, $B \rightarrow A$ and $C \rightarrow B$.

Now, the closure of the dependencies in F_{AB} and F_{BC} includes $C \rightarrow A$ (because, from $C \rightarrow B$, $B \rightarrow A$ and transitivity rule, we compute as $C \rightarrow A$).

Unit – 5 (Transaction Management and Concurrency Control)

Transaction

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Let's take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A's Account

```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

B's Account

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

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 **Atomicity**, **Consistency**, **Isolation**, and **Durability** – commonly known as ACID properties – in order to ensure accuracy, completeness, and data integrity.

- **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.
- **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.
- **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.
- **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.

Transaction Log

- In the field of [databases](#) in [computer science](#), a **transaction log** (also **transaction journal**, **database log**, **binary log** or **audit trail**) is a history of actions executed by a [database management system](#) used to guarantee [ACID](#) properties over [crashes](#) or hardware failures. Physically, a log is a [file](#) listing changes to the database, stored in a stable storage format.
- If, after a start, the database is found in an [inconsistent](#) state or not been shut down properly, the database management system reviews the database logs for [uncommitted](#) transactions and [rolls back](#) the changes made by these [transactions](#). Additionally, all transactions that are already committed but whose changes were not yet materialized in the database are re-applied. Both are done to ensure [atomicity](#) and [durability](#) of transactions.

Anatomy of a general database log

A database log record is made up of:

- *Log Sequence Number* (LSN): A unique ID for a log record. With LSNs, logs can be recovered in constant time. Most LSNs are assigned in monotonically increasing order, which is useful in recovery [algorithms](#), like [ARIES](#).
- *Prev LSN*: A link to their last log record. This implies database logs are constructed in [linked list](#) form.
- *Transaction ID number*: A reference to the database transaction generating the log record.
- *Type*: Describes the type of database log record.
- Information about the actual changes that triggered the log record to be written.

Types of database log records

All log records include the general log attributes above, and also other attributes depending on their type (which is recorded in the *Type* attribute, as above).

- **Update Log Record** notes an update (change) to the database. It includes this extra information:
 - *PageID*: A reference to the Page ID of the modified page.
 - *Length and Offset*: Length in bytes and offset of the page are usually included.
 - *Before and After Images*: Includes the value of the bytes of page before and after the page change. Some databases may have logs which include one or both images.
- **Compensation Log Record** notes the rollback of a particular change to the database. Each corresponds with exactly one other Update Log Record (although the corresponding update log record is not typically stored in the Compensation Log Record). It includes this extra information:
 - *undoNextLSN*: This field contains the LSN of the next log record that is to be undone for transaction that wrote the last Update Log.
- **Commit Record** notes a decision to commit a transaction.
- **Abort Record** notes a decision to abort and hence roll back a transaction.
- **Checkpoint Record** notes that a checkpoint has been made. These are used to speed up recovery. They record information that eliminates the need to read a long way into

the log's past. This varies according to checkpoint algorithm. If all dirty pages are flushed while creating the checkpoint (as in [PostgreSQL](#)), it might contain:

- **redoLSN**: This is a reference to the first log record that corresponds to a dirty page. i.e. the first update that wasn't flushed at checkpoint time. This is where redo must begin on recovery.
- **undoLSN**: This is a reference to the oldest log record of the oldest in-progress transaction. This is the oldest log record needed to undo all in-progress transactions.
- **Completion Record** notes that all work has been done for this particular transaction. (It has been fully committed or aborted)

Transaction Control

The following commands are used to control transactions.

- **COMMIT** – to save the changes.
- **ROLLBACK** – to roll back the changes.
- **SAVEPOINT** – creates points within the groups of transactions in which to ROLLBACK.
- **SET TRANSACTION** – Places a name on a transaction.

Transactional Control Commands

Transactional control commands are only used with the **DML Commands** such as - INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

The COMMIT Command

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for the COMMIT command is as follows.

```
COMMIT;
```

Example

Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00

6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

+-----+-----+-----+-----+-----+

Following is an example which would delete those records from the table which have age = 25 and then COMMIT the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
```

```
WHERE AGE = 25;
```

```
SQL> COMMIT;
```

Thus, two rows from the table would be deleted and the SELECT statement would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

The ROLLBACK Command

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for a ROLLBACK command is as follows –

```
ROLLBACK;
```

Example

Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00

4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

+-----+-----+-----+-----+-----+

Following is an example, which would delete those records from the table which have the age = 25 and then ROLLBACK the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
```

```
WHERE AGE = 25;
```

```
SQL> ROLLBACK;
```

Thus, the delete operation would not impact the table and the SELECT statement would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

The SAVEPOINT Command

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for a SAVEPOINT command is as shown below.

```
SAVEPOINT SAVEPOINT_NAME;
```

This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as shown below.

```
ROLLBACK TO SAVEPOINT_NAME;
```

Following is an example where you plan to delete the three different records from the CUSTOMERS table. You want to create a SAVEPOINT before each delete, so that you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

Example

Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

The following code block contains the series of operations.

```
SQL> SAVEPOINT SP1;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=1;

1 row deleted.

SQL> SAVEPOINT SP2;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=2;

1 row deleted.

SQL> SAVEPOINT SP3;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=3;
```


1 row deleted.

Now that the three deletions have taken place, let us assume that you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone –

```
SQL> ROLLBACK TO SP2;
```

Rollback complete.

Notice that only the first deletion took place since you rolled back to SP2.

```
SQL> SELECT * FROM CUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

6 rows selected.

The RELEASE SAVEPOINT Command

The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created.

The syntax for a RELEASE SAVEPOINT command is as follows.

```
RELEASE SAVEPOINT SAVEPOINT_NAME;
```

Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT.

The SET TRANSACTION Command

The SET TRANSACTION command can be used to initiate a database transaction. This command is used to specify characteristics for the transaction that follows. For example, you can specify a transaction to be read only or read write.

The syntax for a SET TRANSACTION command is as follows.

```
SET TRANSACTION [ READ WRITE | READ ONLY ] ;
```

Concurrency Control

In the concurrency control, the multiple transactions can be executed simultaneously.

It may affect the transaction result. It is highly important to maintain the order of execution of those transactions.

Problems of concurrency control

Several problems can occur when concurrent transactions are executed in an uncontrolled manner. Following are the three problems in concurrency control.

1. Lost updates
2. Dirty read
3. Unrepeatable read

1. Lost update problem

- When two transactions that access the same database items contain their operations in a way that makes the value of some database item incorrect, then the lost update problem occurs.
- If two transactions T1 and T2 read a record and then update it, then the effect of updating of the first record will be overwritten by the second update.

Example:

Transaction-X	Time	Transaction-Y
—	t1	—
Read A	t2	—
—	t3	Read A
Update A	t4	—
—	t5	Update A
—	t6	—

Here,

- At time t2, transaction-X reads A's value.
- At time t3, Transaction-Y reads A's value.
- At time t4, Transactions-X writes A's value on the basis of the value seen at time t2.
- At time t5, Transactions-Y writes A's value on the basis of the value seen at time t3.
- So at time T5, the update of Transaction-X is lost because Transaction y overwrites it without looking at its current value.
- Such type of problem is known as Lost Update Problem as update made by one transaction is lost here.

2. Dirty Read

- The dirty read occurs in the case when one transaction updates an item of the database, and then the transaction fails for some reason. The updated database item is accessed by another transaction before it is changed back to the original value.

- A transaction T1 updates a record which is read by T2. If T1 aborts then T2 now has values which have never formed part of the stable database.

Example:

Transaction-X	Time	Transaction-Y
—	t1	—
—	t2	Update A
Read A	t3	—
—	t4	Rollback
—	t5	—

- At time t2, transaction-Y writes A's value.
- At time t3, Transaction-X reads A's value.
- At time t4, Transactions-Y rollbacks. So, it changes A's value back to that of prior to t1.
- So, Transaction-X now contains a value which has never become part of the stable database.
- Such type of problem is known as Dirty Read Problem, as one transaction reads a dirty value which has not been committed.

3. Inconsistent Retrievals Problem

- Inconsistent Retrievals Problem is also known as unrepeatable read. When a transaction calculates some summary function over a set of data while the other transactions are updating the data, then the Inconsistent Retrievals Problem occurs.
- A transaction T1 reads a record and then does some other processing during which the transaction T2 updates the record. Now when the transaction T1 reads the record, then the new value will be inconsistent with the previous value.
- **Example:** Suppose two transactions operate on three accounts.

Account-1	Account-2	Account-3
Balance = 200	Balance = 250	Balance = 150

Transaction-X	Time	Transaction-Y
—	t1	—
Read Balance of Acc-1 sum <-- 200	t2	—
Read Balance of Acc-2 Sum <-- Sum + 250 = 450	t3	—
—	t4	Read Balance of Acc-3
—	t5	Update Balance of Acc-3 150 --> 150 - 50 --> 100
—	t6	Read Balance of Acc-1
—	t7	Update Balance of Acc-1 200 --> 200 + 50 --> 250
Read Balance of Acc-3 Sum <-- Sum + 250 = 450	t8	COMMIT
—	t9	—

- Transaction-X is doing the sum of all balance while transaction-Y is transferring an amount 50 from Account-1 to Account-3.

- Here, transaction-X produces the result of 550 which is incorrect. If we write this produced result in the database, the database will become an inconsistent state because the actual sum is 600.
- Here, transaction-X has seen an inconsistent state of the database.

Concurrency Control Protocol

Concurrency control protocols ensure atomicity, isolation, and serializability of concurrent transactions. The concurrency control protocol can be divided into three categories:

1. Lock based protocol
2. Time-stamp protocol
3. Validation based protocol

Concurrency Control Problems

The coordination of the simultaneous execution of transactions in a multiuser database system is known as concurrency control. The objective of concurrency control is to ensure the serializability of transactions in a multiuser database environment. Concurrency control is important because the simultaneous execution of transactions over a shared database can create several data integrity and consistency problems. The three main problems are lost updates, uncommitted data, and inconsistent retrievals.

1. Lost Updates:

The lost update problem occurs when two concurrent transactions, T1 and T2, are updating the same data element and one of the updates is lost (overwritten by the other transaction). Consider the following PRODUCT table example.

One of the PRODUCT table's attributes is a product's quantity on hand (PROD_QOH).

Assume that you have a product whose current PROD_QOH value is 35. Also assume that two concurrent transactions, T1 and T2, occur that update the PROD_QOH value for some item in the PRODUCT table.

The transactions are as follows.

Two concurrent transactions update PROD_QOH:

Transaction	Operation
T1: Purchase 100 units	$\text{PROD_QOH} = \text{PROD_QOH} + 100$
T2: Sell 30 units	$\text{PROD_QOH} = \text{PROD_QOH} - 30$

The Following table shows the serial execution of those transactions under normal circumstances, yielding the correct answer $\text{PROD_QOH} = 105$.

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	$\text{PROD_QOH} = 35 + 100$	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	$\text{PROD_QOH} = 135 - 30$	
6	T2	Write PROD_QOH	105

But suppose that a transaction is able to read a product's PROD_QOH value from the table before a previous transaction (using the same product) has been committed.

The sequence depicted in the following Table shows how the lost update problem can arise. Note that the first transaction (T1) has not yet been committed when the second transaction (T2) is executed. Therefore, T2 still operates on the value 35, and its subtraction yields 5 in memory. In the meantime, T1 writes the value 135 to disk, which is promptly overwritten by T2. In short, the addition of 100 units is "lost" during the process.

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	$\text{PROD_QOH} = 35 + 100$	
4	T2	$\text{PROD_QOH} = 35 - 30$	
5	T1	Write PROD_QOH (Lost update)	135
6	T2	Write PROD_QOH	5

2. Uncommitted Data:

The phenomenon of uncommitted data occurs when two transactions, T1 and T2, are executed concurrently and the first transaction (T1) is rolled back after the second transaction (T2) has already accessed the uncommitted data—thus violating the isolation property of transactions.


To illustrate that possibility, let's use the same transactions described during the lost updates discussion. T1 has two atomic parts to it, one of which is the update of the inventory, the other possibly being the update of the invoice total (not shown). T1 is forced to roll back due to an error during the updating of the invoice's total; hence, it rolls back all the way, undoing the inventory update as well. This time, the T1 transaction is rolled back to eliminate the addition of the 100 units. Because T2 subtracts 30 from the original 35 units, the correct answer should be 5.

Transaction	Operation
T1: Purchase 100 units	$\text{PROD_QOH} = \text{PROD_QOH} + 100$ (Rolled back)
T2: Sell 30 units	$\text{PROD_QOH} = \text{PROD_QOH} - 30$

The following Table shows how, under normal circumstances, the serial execution of those transactions yields the correct answer.

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T1	****ROLLBACK****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 - 30	
7	T2	Write PROD_QOH	5

The following Table shows how the uncommitted data problem can arise when the ROLLBACK is completed after T2 has begun its execution.

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	z135
4	T2	Read PROD_QOH (Read uncommitted data) 	135
5	T2	PROD_QOH = 135 - 30	
6	T1	****ROLLBACK****	35
7	T2	Write PROD_QOH	105

3. Inconsistent Retrievals:

Inconsistent retrievals occur when a transaction accesses data before and after another transaction(s) finish working with such data. For example, an inconsistent retrieval would occur if transaction T1 calculated some summary (aggregate) function over a set of data while another transaction (T2) was updating the same data. The problem is that the transaction might read some data before they are changed and other data after they are changed, thereby yielding inconsistent results.

To illustrate that problem, assume the following conditions:

1. T1 calculates the total quantity on hand of the products stored in the PRODUCT table.
2. At the same time, T2 updates the quantity on hand (PROD_QOH) for two of the PRODUCT table's products.

The two transactions are shown in the following Table:

TRANSACTION T1		TRANSACTION T2	
SELECT	SUM(PROD_QOH)	UPDATE	PRODUCT
FROM	PRODUCT	SET	PROD_QOH = PROD_QOH + 10
		WHERE	PROD_CODE = '1546-QQ2'
		UPDATE	PRODUCT
		SET	PROD_QOH = PROD_QOH - 10
		WHERE	PROD_CODE = '1558-QW1'
		COMMIT;	

While T1 calculates the total quantity on hand (PROD_QOH) for all items, T2 represents the correction of a typing error: the user added 10 units to product 1558-QW1's PROD_QOH but meant to add the 10 units to product 1546-QQ2's PROD_QOH. To correct the problem, the user adds 10 to product 1546-QQ2's PROD_QOH and subtracts 10 from product 1558-QW1's PROD_QOH. The initial and final PROD_QOH values are reflected in the following Table

	BEFORE	AFTER
PROD_CODE	PROD_QOH	PROD_QOH
11QER/31	8	8
13-Q2/P2	32	32
1546-QQ2	15	(15 + 10) → 25
1558-QW1	23	(23 - 10) → 13
2232-QTY	8	8
2232-QWE	6	6
Total	92	92

The following table demonstrates that inconsistent retrievals are possible during the transaction execution, making the result of T1's execution incorrect. The "After" summation shown in Table 10.9 reflects the fact that the value of 25 for product 1546-QQ2 was read after the WRITE statement was completed. Therefore, the "After" total is 40 + 25 = 65. The "Before" total reflects the fact that the value of 23 for product 1558-QW1 was read before the next WRITE statement was completed to reflect the corrected update of 13. Therefore, the "Before" total is 65 + 23 = 88.

TIME	TRANSACTION	ACTION	VALUE	TOTAL
1	T1	Read PROD_QOH for PROD_CODE = '11QER/31'	8	8
2	T1	Read PROD_QOH for PROD_CODE = '13-Q2/P2'	32	40
3	T2	Read PROD_QOH for PROD_CODE = '1546-QQ2'	15	
4	T2	PROD_QOH = 15 + 10		
5	T2	Write PROD_QOH for PROD_CODE = '1546-QQ2'	25	
6	T1	Read PROD_QOH for PROD_CODE = '1546-QQ2'	25	(After) 65
7	T1	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	(Before) 88
8	T2	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	
9	T2	PROD_QOH = 23 - 10		
10	T2	Write PROD_QOH for PROD_CODE = '1558-QW1'	13	
11	T2	***** COMMIT *****		
12	T1	Read PROD_QOH for PROD_CODE = '2232-QTY'	8	96
13	T1	Read PROD_QOH for PROD_CODE = '2232-QWE'	6	102

The computed answer of 102 is obviously wrong because you know from the previous Table that the correct answer is 92. Unless the DBMS exercises concurrency control, a multiuser database environment can create havoc within the information system.

The Scheduler and its Functions

You now know that severe problems can arise when two or more concurrent transactions are executed. You also know that a database transaction involves a series of database I/O operations that take the database from one consistent state to another. Finally, you know that database consistency can be ensured only before and after the execution of transactions.

- A database always moves through an unavoidable temporary state of inconsistency during a transaction's execution if such transaction updates multiple tables/rows. (If the transaction contains only one update, then there is no temporary inconsistency.) That temporary inconsistency exists because a computer executes the operations serially, one after another. During this serial process, the isolation property of transactions prevents them from accessing the data not yet released by other transactions.
- The scheduler establishes the order in which the operations with in concurrent transactions are executed. The scheduler interleaves the execution of database operations to ensure serializability. To determine the appropriate order, the scheduler bases its actions on concurrency control algorithms, such as locking or time-stamping methods. The scheduler also makes sure that the computer's CPU is used efficiently.
- The DBMS determines what transactions are serializable and proceeds to interleave the execution of the transaction's operations. Generally, transactions that are not serializable are executed on a first-come, first-served basis by the DBMS. The scheduler's main job is to create a serializable schedule of a transaction's operations.
- A serializable schedule is a schedule of a transaction's operations in which the interleaved execution of the transactions (T1, T2, T3, etc.) yields the same results as if the transactions were executed in serial order (one after another).
- The scheduler also makes sure that the computer's central processing unit (CPU) and storage systems are used efficiently. If there were no way to schedule the execution of transactions, all transactions would be executed on a first-come, first-served basis. The problem with that approach is that processing time is wasted when the CPU waits for a READ or WRITE operation to finish, thereby losing several CPU cycles.
- The scheduler facilitates data isolation to ensure that two transactions do not update the same data element at the same time. Database operations might require READ and/or WRITE actions that produce conflicts. For example, The following Table shows the possible conflict scenarios when two transactions, T1 and T2, are executed concurrently over the same data. Note that in Table 10.11, two operations are in conflict when they access the same data and at least one of them is a WRITE operation.

	TRANSACTIONS		RESULT
	T1	T2	
Operations	Read	Read	No conflict
	Read	Write	Conflict
	Write	Read	Conflict
	Write	Write	Conflict

CONCURRENCY CONTROL WITH LOCKING METHODS

A **lock** guarantees exclusive use of a data item to a current transaction. In other words, transaction T2 does not have access to a data item that is currently being used by transaction T1. A transaction acquires a lock prior to data access; the lock is released (unlocked) when the transaction is completed so that another transaction can lock the data item for its exclusive use.

Most multiuser DBMSs automatically initiate and enforce locking procedures. All lock information is managed by a **lock manager**.

Lock Granularity

Indicates the level of lock use. Locking can take place at the following levels: database, table, page, row or even field.

LOCK TYPES

Regardless of the level of locking, the DBMS may use different lock types:

1. Binary Locks

Have only two states: locked (1) or unlocked (0).

2. Shared/Exclusive Locks

An **exclusive lock** exists when access is reserved specifically for the transaction that locked the object. The exclusive lock must be used when the potential for conflict exists. A **shared lock** exists when concurrent transactions are granted read access on the basis of common lock. A shared lock produces no conflict as long as all the concurrent transactions are read only.

DEADLOCKS

A deadlock occurs when two transactions wait indefinitely for each other to unlock data.

The three basic techniques to control deadlocks are:

- Deadlock prevention . A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur. if the transaction is aborted , all changes made by this transaction are rolled back and all locks obtained by the transaction are released .The transaction is then rescheduled for execution.
- Deadlock detection. The DBMS periodically tests the database for deadlocks. if a deadlock is found one of the transactions is aborted (rolled back and restarted) and the other transaction are continues.
- Deadlock avoidance. The transaction must obtain all of the locks it needs before it can be executed .This technique avoids the rollback of conflicting transactions by requiring that locks be obtained in succession

What is Two-Phase Locking (2PL)?

- Two-Phase Locking (2PL) is a concurrency control method which divides the execution phase of a transaction into three parts.
- It ensures conflict serializable schedules.
- If read and write operations introduce the first unlock operation in the transaction, then it is said to be Two-Phase Locking Protocol.

This protocol can be divided into two phases,

- 1. In Growing Phase,** a transaction obtains locks, but may not release any lock.
- 2. In Shrinking Phase,** a transaction may release locks, but may not obtain any lock.

- Two-Phase Locking does not ensure freedom from deadlocks.

Types of Two – Phase Locking Protocol

Following are the types of two – phase locking protocol:

1. Strict Two – Phase Locking Protocol
2. Rigorous Two – Phase Locking Protocol
3. Conservative Two – Phase Locking Protocol

1. Strict Two-Phase Locking Protocol

- Strict Two-Phase Locking Protocol avoids cascaded rollbacks.
- This protocol not only requires two-phase locking but also all exclusive-locks should be held until the transaction commits or aborts.
- It is not deadlock free.
- It ensures that if data is being modified by one transaction, then other transaction cannot read it until first transaction commits.
- Most of the database systems implement rigorous two – phase locking protocol.

2. Rigorous Two-Phase Locking

- Rigorous Two – Phase Locking Protocol avoids cascading rollbacks.
- This protocol requires that all the share and exclusive locks to be held until the transaction commits.

3. Conservative Two-Phase Locking Protocol

- Conservative Two – Phase Locking Protocol is also called as Static Two – Phase Locking Protocol.
- This protocol is almost free from deadlocks as all required items are listed in advanced.
- It requires locking of all data items to access before the transaction starts.

Timestamp-based Protocols

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

Concurrency control with time stamp ordering

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

- The timestamp of transaction T_i is denoted as $TS(T_i)$.
- Read time-stamp of data-item X is denoted by $R\text{-timestamp}(X)$.
- Write time-stamp of data-item X is denoted by $W\text{-timestamp}(X)$.

Timestamp ordering protocol works as follows –

- **If a transaction T_i issues a read(X) operation –**
 - If $TS(T_i) < W\text{-timestamp}(X)$
 - Operation rejected.
 - If $TS(T_i) \geq W\text{-timestamp}(X)$
 - Operation executed.
 - All data-item timestamps updated.
- **If a transaction T_i issues a write(X) operation –**
 - If $TS(T_i) < R\text{-timestamp}(X)$
 - Operation rejected.
 - If $TS(T_i) < W\text{-timestamp}(X)$
 - Operation rejected and T_i rolled back.
 - Otherwise, operation executed.

Thomas' Write Rule

This rule states if $TS(T_i) < W\text{-timestamp}(X)$, then the operation is rejected and T_i is rolled back.

Time-stamp ordering rules can be modified to make the schedule view serializable.

Instead of making T_i rolled back, the 'write' operation itself is ignored.

Deadlock

In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

For example, assume a set of transactions $\{T_0, T_1, T_2, \dots, T_n\}$. T_0 needs a resource X to complete its task. Resource X is held by T_1 , and T_1 is waiting for a resource Y , which is held by T_2 . T_2 is waiting for resource Z , which is held by T_0 . Thus, all the processes wait for each other to release resources. In this situation, none of the processes can finish their task. This situation is known as a deadlock.

Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

Deadlock Prevention

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

Wait-Die Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur –

- If $TS(T_i) < TS(T_j)$ – that is T_i , which is requesting a conflicting lock, is older than T_j – then T_i is allowed to wait until the data-item is available.
- If $TS(T_i) > TS(T_j)$ – that is T_i is younger than T_j – then T_i dies. T_i is restarted later with a random delay but with the same timestamp.

This scheme allows the older transaction to wait but kills the younger one.

Wound-Wait Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some another transaction, one of the two possibilities may occur –

- If $TS(T_i) < TS(T_j)$, then T_i forces T_j to be rolled back – that is T_i wounds T_j . T_j is restarted later with a random delay but with the same timestamp.
- If $TS(T_i) > TS(T_j)$, then T_i is forced to wait until the resource is available.

This scheme, allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.

In both the cases, the transaction that enters the system at a later stage is aborted.

Deadlock Avoidance

Aborting a transaction is not always a practical approach. Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance. Methods like "wait-for graph" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource. In a bulky system, deadlock prevention techniques may work well.

Crash Recovery

DBMS is a highly complex system with hundreds of transactions being executed every second. The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

Failure Classification

To see where the problem has occurred, we generalize a failure into various categories, as follows –

Transaction failure

A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further. This is called transaction failure where only a few transactions or processes are hurt.

Reasons for a transaction failure could be –

- **Logical errors** – Where a transaction cannot complete because it has some code error or any internal error condition.
- **System errors** – Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash

There are problems – external to the system – that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.

Examples may include operating system errors.

Disk Failure

In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

Storage Structure

We have already described the storage system. In brief, the storage structure can be divided into two categories –

- **Volatile storage** – As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
- **Non-volatile storage** – These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks, magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following –

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction –

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Log-based Recovery

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows –

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.

<T_n, Start>

- When the transaction modifies an item X, it write logs as follows –

<T_n, X, V₁, V₂>

It reads T_n has changed the value of X, from V₁ to V₂.

- When the transaction finishes, it logs –

<T_n, commit>

The database can be modified using two approaches –

- **Deferred database modification** – All logs are written on to the stable storage and the database is updated when a transaction commits.
- **Immediate database modification** – Each log follows an actual database modification. That is, the database is modified immediately after every operation.

Recovery with Concurrent Transactions

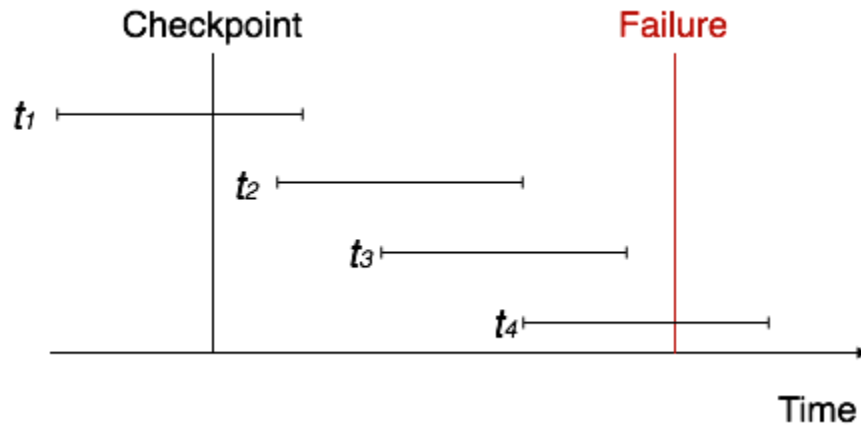
When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

Checkpoint

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner –



- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$ or just $\langle T_n, \text{Commit} \rangle$, it puts the transaction in the redo-list.
- If the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ but no commit or abort log found, it puts the transaction in undo-list.

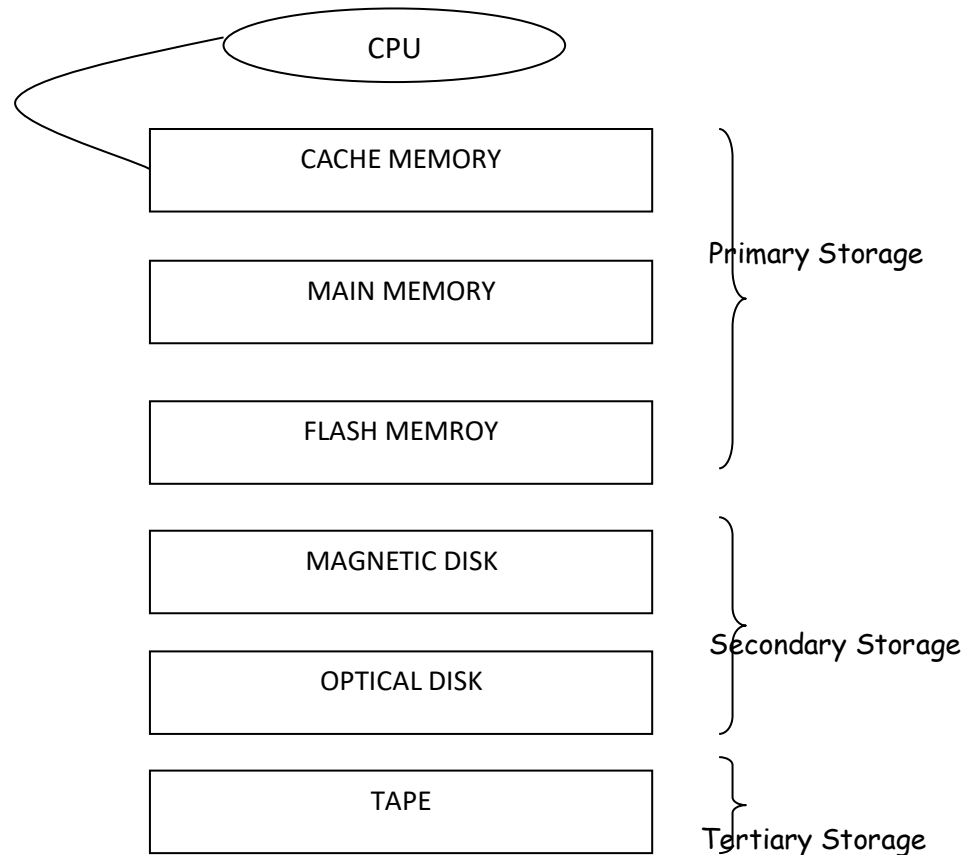
All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

UNIT – 6 :: Storage and Indexing

Syllabus: Data on External Storage, File Organization and Indexing, Cluster Indexes, Primary and Secondary Indexes, Index data Structures, hash Based Indexing: Tree base Indexing, Comparison of File Organizations, Indexes and Performance Tuning.

Data on External Storage: The data stored on the database is very large that cannot accommodate in main memory and cannot store permanently. To store large volume of data permanently, an external storage devices were developed.

Different Kind of memory in a Computer System (or) Memory Hierarchy: Memory in a computer system is arranged in a hierarchy is shown as follows.



→ At the top, there is Primary storage that has cache, flash and main memory to provide very fast access the data.

→ The secondary storage devices are Magnetic disks that are slower and permanent devices.

→ The Tertiary Storage is a permanent and slowest device when compared with Magnetic disk.

Cache memory: The cache is the fastest but costliest memory available. It is not concern for databases.

Main Memory: The processor requires the data to be stored in main memory. Although main memory contains Giga byte of storage capacity but it is not sufficient for databases.

Flash Memory: Flash memory stores data even if the power fails. Data can be retrieved as fast as in main memory, however writing data to flash memory is a complex task and overwriting data cannot be done directly. It is used in small computers.

Magnetic Disk Storage: Magnetic disk is the permanent data storage medium. It enables random access of data and it is called "Direct-access" storage. Data from disk is transferred into main memory for processing. After modification, the data is loaded back onto the disk.

Optical Disk: Optical disks are Compact Disks (CDs), Digital Video Disks (DVDs). These are commonly used for permanent data storage. CDs are used for providing electronically published information and for distributing software such as multimedia data. They have a larger capacity that is upto 640 MB. These are relatively cheaper. To storage large volume of data CDs are replaced with DVDs. DVDs are in various capacities based on manufacturing.

Advantages:

1. Optical disks are less expensive.
2. Large amount of data can be stored.
3. CDs and DVDs are longer durability than magnetic disk drives.
4. These provide nonvolatile storage of data.
5. These can store any type of data such text data, music, video etc.

Tape Storage (or) Tertiary Storage media: Tape (or) Tertiary storage provides only sequential access to the data and the access to data is much slower. They provide high capacity removable tapes. They can have capacity of about 20 Giga bytes to 40 GB. These devices are also called "tertiary storage" or "off the storage". In a larger database system, tape (tertiary) storage devices are using for backup storage of data.

Magnetic tapes are fragmented into vertical columns referred as frames and horizontal rows referred as tracks. The data is organized in the form of column string with one data/frame. Frames are in turn fragmented into rows or tracks. One frame can store one byte of data and individual track can store a single bit. The rest of the tract is treated as a parity track.

Advantages:

1. Magnetic tapes are very less expensive and durable compared with optical disks.
2. These are reliable and a good tape drive system performs a read/write operation successfully.
3. These are very good choice for archival storage and any number of times data can be erased and reused.

Disadvantages:

The major disadvantage of tapes is that they are sequential access devices. They work very slow when compared to magnetic disks and optical disks.

Performance Implications of Disk Structure:

1. Data must be in memory for the the DBMS to operate on it.
2. The unit for data transfer between disk and main memory is a block;if a single item on a block is needed,the entire block is transferred.Reading or writing a disk block is called an I/O(for input/output)operation.
3. The time to read or Write a block varies, depending on the location of the location of the data:
 $\text{Access time} = \text{seek time} + \text{rotational delay} + \text{transfer time}$
- 4.the time for moving blocks to or from disk usually dominates the time taken for database operations.To minimize this time, it is necessary to locate data records strategically on disk because of the geometry and mechanics of disks.

Buffer Manager: The buffer manager is the software layer that is responsible for bringing pages from physical disk to main memory as needed. The buffer manager manages the available main memory by dividing into a collection of pages, which we called as buffer pool. The main memory pages in the buffer pool are called frames.

The goal of the buffer manager is to ensure that the data requests made by programs are satisfied by copying data from secondary storage devices into buffer. In fact, if a program performs an input statement, it calls the buffer manager for input operation to satisfy the requests by reading from existing buffers.

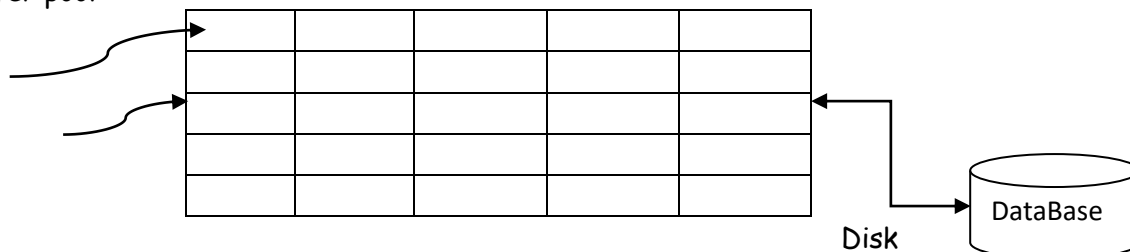
Similarly, if a program performs an output statement, it calls the buffer manager for output operation to satisfy the requests by writing to the buffers. Therefore, we can say that input and output operations occurs b/w the program and the buffer area only.

In addition to the buffer pool itself, the buffer manager maintains two variables for each frame in the pool. They are '**pin-count**' and '**dirty**'. The number of times the page is requested in the frame, each time the pin-count variable is incremented for that frame (i.e. set to 1 (pin-count=1)). For satisfying each request the pin-count is decreased for that frame (i.e. set to 0).

Thus, if a page is requested the pin-count is incremented, if it fulfills the request the pin-count is decremented.

In addition to this, if the page has been modified the Boolean variable, 'dirty' is set as 'on'. Otherwise set to 'off'.

Buffer pool



→ **Buffer Manager Writing the page to disk:** When a page is requested the buffer manager does the following.

1. Checks the buffer pool that frame contains the requested page and if so, increment the pin-count of that frame. If the page is not in the pool, the buffer manager brings it into the main memory from disk and set the pin-count value as 1. Otherwise set to 0.
2. If the 'dirty' variable is set to 'on' then that page is modified and replaced by its previous page and writes that page on to the disk.

→ **Pinning and Unpinning of pages:** In buffer pool if some frame contains the requested page, the pin-count variable is set to 1 of that frame. So, pin-count = 1 is called **pinning** the requested page in its frame. When the request of the requestor is fulfilled, the pin-count variable is set to 0 of that frame. Thus, the buffer manager will not read any page into a frame unit when pin-count becomes 0 (zero).

→ **Allocation of Records to Blocks:** The buffer management uses block of storage and it is replaced by next record allocation when current record is deleted. The buffer manager also provides concurrency control system to execute more than one process. In this case, the records are mapped onto disk blocks.

File Organization and Indexing:

File Organization: A file organization is a method of logically arranging the records in file on the disk. These records are mapped onto disk blocks.

In DBMS, the file of records is an important. That is, create a file, destroy it and also can insert records into it, delete from it. It supports scan also. The most important and widely used storage technique is Heap file. A Heap file is the simplest file structure. In a heap file, records are stored in random order across the pages of the file.

Thus, a file organization can be defined as the process of arranging the records in a file, when the file is stored on disk.

Types of File Organizations: Data is organized on secondary storage in terms of files. Each file has several records.

The enormous data cannot be stored in main memory so, it is stored on magnetic disk. During the process, the required data can be shifted to main memory from disk. The unit of information being transferred b/w main memory and disk is called a page.

Tapes are also used to store the data in the database. But this can be accessed sequentially. So, most of the time is wasted to transfer each page.

Buffer manager is software used for reading data into memory and writing data onto magnetic disks. Each record on a file is identified by record id or rid. Whenever a page needs to be processed, the buffer manager retrieves the page from the disk based on its record id.

Disk space manager is software that allocates space for records on the disk. When DBMS requires an additional space, it calls disk space management to allocate the space. DBMS also informs disk space manager when it's not going to use the space.

Most widely used file organizations are 1. Heap (Unordered) Files. 2. Sequential (ordered) files. 3. Hash files.

1. **Heap file:** It is the simplest file organization and stores the files in the order they arrive. It also called unordered file.

→**Inserting a record:** Records are inserted in the same order as they arrive.

→**Deleting a record:** The record is to be deleted, first access that record and then marked as deleted.

→**Accessing a Record:** A linear search is performed on the files starting from the first record until the desired record is found.

2. **Ordered File:** Files are arranged in sequential order. The main advantage of this file organization is that we can now use binary search as the file are sorted.

→**Insertion of Record:** This is a difficult task. Because, first we need to identify the space where record need insert and file is arranged in an order. If the space is available then record can directly be inserted. If space is not sufficient, then that record moved to next page.

→ **Deletion of Files:** This task is also difficult to delete the record. In this first find deleted record and then remove the empty space of deleted record.

→**Access of files:** This is similar as we can use binary search on files.

3. **Hash files:** Using this file organization, files are not organized sequentially, instead they are arranged randomly. The address of the page where the record is to be stored is calculated using a 'hash function'.

Index: An index is a data structure which organizes data records on disk to optimize certain kinds of retrieval operations. Using an index, we easily retrieve the records which satisfy search conditions on the search key fields of the index. The 'data entry' is the term which we use to refer to the records stored in an index file. We can search an index efficiently for finding desired data entries and use them for obtaining data records. There are three alternatives for to store a data in an index,

1) A data entry K^* is the actual data record with search key value of k

2) A data entry is a $\langle k, rid \rangle$ pair (Here rid is the row id or record id and k is key value).

3) A data entry is a $\langle k, rid-list \rangle$ pair (rid-list is a list of record ids of the data records with search key value k).

Types of Index: There two index techniques to organize the file. They are 1. Clustered. 2. Un-clustered.

Clustered: A file organization in which the data records are ordered in same way as data entries in index is called clustered.

Un-clustered: A file organization in which the data records are ordered in a different way as data entries in index is called un-clustered.

Indexed Sequential files : Indexed sequential file overcomes the disadvantages of sequential file, where in it is not possible to directly access a particular record. But, in index sequential file organization, it is possible to access the record both sequentially and randomly. Similar to sequential file, the records in indexed sequential file are organized in sequence based on primary key values. In addition to this, indexed sequential file consists of the following two features that distinguishes it from a sequential file.

i)**Index:** It is used so as to support random access. It provides a lookup capability to reach quickly to the desired record.

ii)**Overflow file:** The overflow file is similar to the log file used in the sequential file. Indexed sequential file greatly reduces the time required to access a single record without sacrificing the sequential nature of the file. In order to process a file sequentially, the records of the main file are initially processed in a sequence until a pointer to the overflow file is found. Then accessing continues in the overflow file until a null pointer is encountered.

Hash File Organization: Hash file organization helps us to locate records very fast with a given search key value. For example, "Find the sailor record for "SAM", if the file is hashed on name field. In hashed files, the pages are grouped into bucket. Every bucket has bucket number which allows us to find the primary page for that bucket. Then the record belonging to that bucket can be determined by using hash function to the search fields, when inserting the record into the appropriate bucket with overflow pages are maintained in a linked list. For searching a record with a given search key value, apply the hash function to identify the bucket to which such records belongs and look at. This organization is called static hashed file.

Fixed-Length File Organization: A file is organized logically as a sequence of records. These records are mapped onto disk block.

One approach to mapping the database to files is to use several files and to store records of only one fixed length in any given file. An alternative is to structure our files so that we can accommodate multiple lengths for records, however, files of fixed length records are easier to implement than the files of variable-length records.

Fixed-length Records: As an example, consider a file of account records for our bank database. Each record of this file is defined as

Type deposit = record

Account_number : char(10);
Branch_name : char(20);
Balance : real;

End

→ In the above definition, each character occupies 1 byte and a real occupies 8 bytes. So totally the Above record occupies 38 bytes long.

Record 1
Record 2
Record 3
Record 4
Record 5
Record 6
Record 7

Acc.no	branch name	balance
101	ongole	2000
205	chimakurthy	3500
301	kandukur	4300
102	ongole	2200
222	chimakurthy	1200
333	kandukur	2600
343	kandukur	2200

→ There is a problem to delete a record from this structure. The space occupied by the record to be deleted must be filled with some other record of the file (or) we must have a way of marking deleted records so that they can be ignored. This is shown in fig.


Record 1
www.Jntufastupdates.com

Acc.no	branch name	balance
101	ongole	2000
301	kandukur	4300
102	ongole	2200
222	chimakurthy	1200

The main disadvantages of this is when one record to be deleted then entire database need to modify why because after deletion of record that space is occupied by its next record and so on.

Record 3
Record 4
Record 5
Record 6
Record 7

To rectify this drawback, DBMS provided a facility that deleted record space is replaced by last record. This is shown in fig.



	Acc.no	branch name	balance
Record 1	101	ongole	2000
Record 7	343	kandukur	2200
Record 3	301	kandukur	4300
Record 4	102	ongole	2200
Record 5	222	chimakurthy	1200
Record 6	333	kandukur	2600

On insertion of a new record, we use the record pointer by the header. We change the head pointer to point to the next available record. If no space is available, we add the new record to the end of the file.

Thus, Insertion and deletion for files of fixed-length records are simple to implement, because the space made available by a deleted record is exactly the space needed to insert a record.

Byte-String Representation: A simple method for implementing variable-length records is to attach a special end-of-record () symbol to the end of each record. We can then store each record as a string of consecutive bytes. This is shown in fig.

Acc name	accno	amt	aacno	amt	accno	amt	
suman	1001	4000	2001	9000	2310	7000	
Sreenivaas	3001	5000					
Kalyan	1201	2300					
Kiran kumara	3021	5000	3233	2200			
Sowjanya	3201	5500					
Sravani	2301	2500					

The byte-string representation has some disadvantages:

- 1) It is not easy to reuse space occupied by a deleted record.
- 2) There is no space, for records to grow longer. If a variable-length record becomes longer, it must be moved, movement is costly if pointers to the records are stored elsewhere in the database, since the pointers must be located and updated.

Thus, the basic byte-string representation described here not usually used for implementing variable-length records. However, a modified form of the byte-string representation, called the slotted-page structure, is commonly used for organizing records within a single block. This is shown in following structure.

size

# Entries							
							6

Location

→ There is a header at the beginning of each block, containing following information,

- 1) The header contains number of record entries.
- 2) The end of free space in the block
- 3) In an array, entries contain the location and size of each record.

The actual records are allocated contiguously in the block, starting from the end of the block. The free space in the block is contiguous, b/w the final entry in the header array and the first records. If a record is inserted, space is allocated for it at the end of free space and an entry containing its size and location is added to the header.

→ If a record is deleted, the space that it occupies is freed and its entry is set to delete. When a record is to be deleted then space is replaced by final entry record.

Fixed-Length Representation: The Fixed-Length representation is another way to implement variable-length records efficiently in a file system to use one or more fixed-length records.

There are two ways to do this, 1) Reserved Space 2) List Representation

1) Reserved Space: If there is a maximum record length that is never exceeded, we can use fixed-length records of that length. Unused space is filled with a special null, or end-of-record, symbol. This is shown in following figure (1).

2) List Representation: We can represent variable-length record by lists of fixed-length records, chained together by pointer. This is shown in fig(2).

fig(1)

suman	1001	4000				
ramu	2201	4400				
Mamatha	3051	3400				
Sumathi	4011	2400				
swapna	3331	4200				
kalyan	6001	4000				

2)

Suman	1001	4000	
Ramu	2201	4400	
Mamatha	3051	3400	
Kalyan	6001	4000	

- The **reserved-space** method is useful when most records have a length close to the maximum. Otherwise, a specified amount of space may be wasted.
-
- The **List Representation** method is useful when an account of bank customer has more accounts in other branches. So, this method add a pointer field to represent the file when an accountant having more accounts in different branches.

→ The disadvantage of the structure is that we waste space in all records except the first in a chain. The first method needs to have the branch_name, but subsequent records do not.

- To overcome this drawback, it allows two kind of blocks in file.

→ **Anchor-block:** Which contain the first records of a chain.

→ **Overflow-block:** Which contain records other than those that are the first records of a chain.

Thus, all records within a block have the same length, even though not all records in the file have the same length.

Types of Indexing: Indexes can perform the DBMS. By the index can locate the desired record directly without scanning each record in the file.

An index can be defined as a data structure that allows faster retrieval of data. Each index is based on certain attribute of the field. This is given in 'search key'.

An index can refer the data based on several search keys. It means, the term data entry refer to the records stored in an index file. The data entry can be a,

1) Search Key. 2) Search key with record id. 3) Search key with list of record ids).

- A file organization when the records are stored and referred in same way as data entries in index is called "clustered".
- A file organization when the records are stored and referred in a different way as data entries in index is called "un-clustered".

Differences b/w clustered and un-clustered:

→ A file organization when the records are stored and referred in same way as data entries in index is called clustered. Whereas un-clustered referred refer in a different way as data entries in index is called un-clustered.

→ A clustered index is an index which uses alternative (1) whereas un-clustered index uses alternative (2) and (3).

→ Clustered index refer few pages when we require retrieving the records. Whereas un-clustered index refer several pages when we require retrieving the records.

→ If a file contains the records in sequential order, the index search key specifies the same order to retrieve the records from the sequential order of the file is called clustered index. Whereas the index search key specifies the different order to retrieve the records from the sequential order of the file is called un-clustered index.

Primary Index and Secondary Index:

Primary Index: A primary index is an index on a set of fields that includes the primary key is called a primary index. A primary index is an ordered file whose records are of fixed length with two fields. The first field of the record of file must be defined with a constraint "primary key" is called the primary key of the data file and the second field is a pointer to a disk block.

There is one index entry in the index file for each block in the data file. Each index entry has the value of the primary key field for the first record in a block and a pointer of that block as its two field values. The two field values of index entry are referred to as key[i], primary key[i].

Primary indexes are further divided into **dense index** and **sparse index**.

1) **Dense Index:** An index record appears for every search key value in the file. The index record contains the search-key value and a pointer to the first record with that search-key.

2) Sparse Index: An index record is created for only some of the values. As it is true in dense indexes, each index record contains a search-key value and a pointer to the first data record with the largest search-key value that is less than or equal to the search-key value for which we are looking. We start at the record pointed to by that index entry and follow the pointers in the file, until we find the desired record.

Secondary Index: An index that is not a primary key index is called a secondary index. That is an index on a set of fields that does not include the primary key is called a secondary index. A secondary index on a candidate key looks just like a dense primary index, except that the records pointed to successive values in the index are not stored sequentially. In general, secondary indices are different from primary indices. If the search key of a primary index is not a primary key, it suffices the index pointing to the first record with a particular value for the search key, since the other records can be fetched by a sequential scan of the file.

A secondary index must contain pointers to all the records, because if the search key of a secondary index is not a primary key, it is not enough to point to just the first record.

Thus, the records are ordered by the search key of the primary index but same search-key value could be anywhere in the file.

4000	
3400	
2240	
4500	
4050	
4200	
4500	
4050	

kalyan	1001	4000
jaishnav	1022	3400
priyanka	2301	2240
kishore	1001	4500
sumahi	4001	4050
chaitanya	3030	4200
Anjali	1001	4500
swapna	4001	4050
mamatha	3030	4200

The above fig. shows the structure of a secondary index that uses an extra level of indirection on the account file, on the search key balance.

A sequential scan in primary index is efficient because records in the file are stored physically in the same order as the index order. We cannot store a file physically ordered both by the search key of the primary index and the search key of a secondary index. Because secondary-key order and physical-key order differ, but if we attempt to scan the file sequentially in secondary-key order, the reading of each record is likely to require the reading of a new block from disk. If a secondary index stores only some of the search key values, records with intermediate search key values may be anywhere in the file and in general, we cannot find them without searching the entire file. Secondary indices must therefore be dense, with an index entry for every search-key value and a pointer to every record in the file but not the sparse.

Secondary indices improve the performance of queries that use keys other than the search key of the primary index. They also impose a significant overhead on modification of the database. The design of a database decides which secondary indices are desirable on the basis of an estimate of the relative frequency of queries and modifications.

Index Data Structures: The two methods in which file data entries can be organized in two ways.

1) Hash-based indexing, which uses search key

- 2) Tree-based indexing, it refers to the process of
- Finding a particular record in a file using one or more index or indexes.
 - Strong a record in any order (randomly on the disk).

1) Hash Based Indexing: This type of indexing is used to find records quickly, by providing a search key value.

In this, a group of file records stored in pages based on bucket method. The first bucket contains a primary page and along with other pages is chained together. In order to determine the bucket for a record, a special function is called a hash function along with a search key is used. By providing a bucket number, we can obtain the primary page in one or more disk I/O operations.

Records Insertion into the Bucket: The records are inserted into the bucket by assigning (allocating) the need "overflow" pages.

Record Searching: a hash function is used to find first, the bucket containing the records and then by scanning all the pages in a bucket, the record with a given search key can be found.

Suppose, if the record doesn't have search key value then all the pages in the file needs to be scanned.

Record Retrieval: By applying a hash function to the record's search key, the page containing the needed record can be identified and retrieved in one disk I/O.

Consider a file student with a hash key rno. Applying the hash function to the rno, represents the page that contains the needed record. The hash function 'h' uses the last two digits of the binary value of the rno as the bucket identifier. A search key index of marks obtained i.e., marks contains <mrks, rid> pairs as data entries in an auxiliary index file which is shown in the fig. The rid (record id) points to the record whose search key value is mrks.

2) Tree-based indexing: In Tree Based indexing the records arranged in tree-like structure. The data entries are started according to the search key values and they are arranged in a hierarchical number to find the correct page of the data entries.

Examples:

- 1) Consider the student record with a search key rno arranged in a tree-structured index. In order to retrieve the nodes (A'_1 , B'_1 , L'_{11} , L'_{12} and L'_{13}) that need to perform disk I/O.

The lowest leaf level contains these records. The additional records with rno's < 19 and > 42 are added to the left side of the leaf node L'_{11} and to the right of the leaf node L'_{13} .

The root node is responsible for the start of search and these searches are then directed to the correct leaf pages by the non-leaf pages which contain node pointers separated by the search key values. The data entries in a subtree smaller than the key value k_i are pointed to by the right node pointer of k_i , shown in fig.

2) In order to find the students whose roll numbers lies b/w '19' and '24', the direction of the search is shown in the fig.

Suppose we want to find all the students roll numbers lying b/w 17 and 40, we first direct the search to the node A'_1 and after analyzing its contents, then forwarded the search to B'_1 followed by the leaf node L'_{11} , which actually contains the required data entry. The other leaf nodes L'_{12} and L'_{13} also contains the data entries that fulfill our search criteria. For this, all the leaf pages must be designed using double linked list.

Thus, L'_{12} can be fetched using next pointer on L'_{11} and L'_{13} can be obtained using the next pointer on L'_{12} . Number of disk I/Os = Length of the path from the root to a leaf (occurs in search) + The number of satisfying data entries leaf pages.

Closed and Open Hashing: File organization based on the technique of hashing allows us to avoid accessing an index structures. Hashing also provides a way of constructing indexes. There are two types of hashing techniques. They are,

1) Static/ Open hashing. 2) Dynamic/Closed hashing.

In a hash file organization, we obtain the address of the disk block containing a desired record directly by computing a function on the search key value of the record. In our description of hashing, we shall use the term bucket to denote a unit of storage that can store one or more records. A bucket is typically a disk block, but could be chosen to be smaller or larger than a disk block.

Static hashing can obtain the address of the disk block containing a desired record directly by computing hash function on the search key value of the record. In static hashing, the number buckets is static (fixed). The static hashing scheme is illustrated as shown in fig.

The pages containing the index data can be viewed as a collection of buckets, with one page and possible additional overflow pages for overflow buckets. A file consists of buckets 0 through $N - 1$ for N buckets. Buckets contain data entries which can be any of the three choices K^* , $\langle k, rid \rangle$ pair, $\langle k, rid-list \rangle$ pair.

To search for a data entry, we apply a hash function 'h' to identify the bucket to which it belongs and then search this bucket.

To insert a data entry, we use the hash function to identify the correct bucket and then put the data entry there. If there is no space for this data entry, we allocate a new overflow bucket, put the data entry and add to the overflow page.

To delete a data entry, we use the hash function to identify the correct bucket, locate the data entry by searching the bucket and then remove it. If this data entry is the last in an overflow page, the overflow page is removed and added to a list of free pages.

Thus, the number of buckets in a static hashing file is known when the file is created the pages can be stored as successive disk pages.

Drawbacks of Static Hashing:

- The main problem with static hashing is that the number of buckets is fixed.
- If a file shrinks greatly, a lot of space is wasted.
- If a file grows a lot, long overflow chains develop, resulting in poor performance.

Dynamic Hashing: The dynamic hashing technique allow the hash function to be modified dynamically to accommodate the growth or shrinkage of the database, because most databases grow larger over time and static hashing techniques presents serious problems to deal with them.

Thus, if we are using static hashing on such growing databases, we have three options:

- 1) Choose a hash function based on the current file size. This option will result in performance degradation as the database grows.
- 2) Choose a hash function based on predicted size of the file for future. This option will result in the wastage of space.
- 3) Periodically reorganize the hash structure in response to file growth.

Thus, using dynamic hashing techniques is the best solution. They are two types,

1. **Extensible Hashing Scheme:** Uses a directory to support inserts and deletes efficiently with no overflow pages.
2. **Linear Hashing Scheme:** Uses a clever policy for creating new buckets and supports inserts and deletes efficiently without the use of a directory.

Comparison of File Organization: To compare file organizations, we consider the following operations that can be performed on a record. They are,

- 1) **Record Insertion:** For inserting a record we need to identify and fetch the page from the disk. The record is then added and the (modified) page is written back to the disk.
- 2) **Record Deletion:** It follows the same procedure as record insertion except that after identifying and fetching a page, the record with the given rid is deleted and again the changed page is added back to the disk.
- 3) **Record Scanning:** In this all the file pages must be fetched from the disk and are stored in a pool of buffers. Then the corresponding record can be retrieved.
- 4) **Record Searching Based on Equality Selection:** In this, all the records that satisfies a given equality selection criteria are fetched from the disk.

Example: To find a student record based on the following equality selection criteria "student whose roll number (rno) is 15 and whose marks (mrks) are 90* is the topper of the class.

- 5) **Record Searching Based on Selected Range:** In this, all the records that satisfies a given equality selection are fetched.

Example: Find all the records of the students whose secured marks are greater than 50.

Cost Model in terms of time needed for execution:

It is a method used to calculate the costs of different operations that are performed on the database.

Notations:

B = The total number of pages without any space wastage when records are grouped into it.

R = The total number of records present in a page.

D = The average time needed to R/W (Read/Write) a disk page.

C = The average time needed for a record processing

H = Time required to apply the hash function on a record in hashed-file organization.

F = Fan-out (in tree indexes)

For calculating I/O costs (which is the base for costs of the database operations) we take,

$D = 15 \text{ ms}$, C and $H = 100 \text{ ns}$.

Heap Files:

1) **Cost of Scanning:** The cost of scanning heap files is given by $B(D + RC)$. It means, Scanning R records of B pages with time C per record would take BRC and scanning B pages with time D per page would take BD . Therefore the total cost of scanning is $BD + BRC \rightarrow B(D + RC)$

2) **Cost of Insertion:** The cost to insert a record in heap file is given as $2D + C$. It means, to insert a record, first we need to fetch the last page of the file that can take time 'D' then we need to add the

record that takes time 'C' and finally the page is written back to the disk from main memory will take time 'D'. So, the total cost is, $D + D + C \rightarrow 2D + C$.

3) Cost of Deletion: The cost to delete a record from a heap file is given as, $D + C + D = 2D + C$. It means, in order to delete a record, first search the record by reading the page that can take time

4) Record Searching based on some quality criteria: Searching exactly one record that meet the equality that involves scanning half of the files based on the assumption to find the record.

This takes time = $\frac{1}{2} \times \text{scanning cost} \rightarrow \frac{1}{2} \times B(D + RC)$.

In case of multiple records the entire file need to be scanned.

5) Record searching with a Range selection: It is the same as the cost of scanning because it is not known in advance how many records can satisfy the particular range. Thus, we need to scan the entire file that would take $B(D + RC)$.

Sorted Files:

1) Cost of scanning: The cost of scanning sorted files is given by $B(D + RC)$ because all the pages need to be scanned in order to retrieve a record. i.e. cost of scanning sorted files = Cost of scanning heap files.

2) Cost Insertion: The cost of insertion in sorted files is given by search cost + $B(D + RC)$.

It includes, Finding correct position of the record + Adding of record + Fetching of pages + rewriting the pages.

3) Cost of Deletion: The cost of deletion in sorted files is given by

Search cost + $B(D + RC)$.

It includes, Searching a record + removing record + rewriting the modified page.

Note: The record deletion is based on equality.

4) Cost of Searching with equality selection Criteria: This cost of sorted files is equal to $D \log_2 B$ = It is the time required for performing a binary search for a page that contain the records.

If many records satisfy, then record is equal to, $D \log_2 B + C \log_2 R$ + Cost of sequential reading of all the records.

5) Cost of Searching with Range Selection: This cost is given as,

Cost of fetching the first matching record's page + Cost of obtaining the set of qualifying records.

If the range is small, then a single page contain all the matching records, else additional pages needs to be fetched.

Clustered Files:

1) Cost of Scanning: The cost of scanning clustered files is same as the cost of scanning sorted files except that it has more number of pages and this cost is given as scanning B pages with time 'D' per page takes BD and scanning R records of B pages with time C per record takes BRC. Therefore the total cost is, $1.5B(D + RC)$.

2) Cost of Insertion: The cost of insertion in clustered files is, Search + Write $(D \log_{1.5} B + C \log_2 R) + D$.

3) Cost of Deletion: It is same as the cost of insertion and includes,

the cost of searching for a record + removing of a record + rewriting the modified page.

i.e. $D \log_{1.5} B + C \log_2 R + D$

4) Equality Selection Search:

i) **For a single Qualifying Record:** The cost of finding a single qualifying record in clustered files is the sum of the binary searches involved in finding the first page in $D \log_{1.5} B$ and finding the first matching record in $C \log_2 R$. i.e. $D \log_{1.5} B + C \log_2 R$.

ii) **For several Qualifying Records:** If more than one record satisfies the selection criteria then they are assumed to be located consecutively.

Cost required to find record is equal to,
 $D \log_F 1.5B + C \log_2 R$ + cost involved in sequential reading of all records.

5) Range Selection Search: This cost in an equality search under several matched records.

Heap File with Un-clustered Tree Index:

1) Scanning: For scanning a student's file,

- i) Scan the index's leaf level.
- ii) Get the relevant record from the file for each data entry.
- iii) Obtain sorted data records according to $\langle rno, mrks \rangle$.

The cost of reading all the data entries is $0.15B(D + 6.7RC)$ I/Os. For each index entry a record has to be fetched in one I/O.

2) Insertion: The record is first inserted at $2D + C$ in students heap file and the associated entry in the index. The correct leaf page can be found in $D \log_F 0.15B + C \log_2 6.7 R$ followed by the addition of a new entry and rewriting in D .

3) Deletion: The cost of deletion includes,

Cost of finding the record in a file + cost of finding the entry in index + Cost of rewriting the modified page in the index and the file.

It corresponds, $D \log_F 0.15B + C \log_2 6.7R + D + 2D$.

4) Equality Selection Search: The cost involved in this operation is the sum of,

- i) The cost of finding the page containing a matched entry.
- ii) The cost of finding the first matched entry and
- iii) The cost of finding the first matched record.

It is given as, $D \log_F 0.15B + C \log_2 6.7R + D$

5) Range Selection Based-search: This is same as search with range selection in clustered files except from having data pages it has data entries.

Heap file with Un-clustered Hash Index:

1) Scan : The total cost is the sum of the cost in the retrieval of all data entries and one I/O cost for each data record. It is given as, $0.125B(D + BRC) + BR(D + C)$.

2) Insertion: It involves the cost of inserting a record i.e., $2D + C$ in the heap file, the cost of finding the page cost of adding a new entry and rewriting of the page, it is expressed as,
 $2D + C + (H + 2D + C)$.

3) Deletion: It involves the cost of finding the data record and the data entry at $H + 2D + 4RC$ and writing back the changed page to the index and file at $2D$. The total cost is, $(H + 2D + 4RC) + 2D$.

4) Equality Selection Search: The total cost in the search accounts to,

- i) The page containing the qualifying entries is identified at the cost H .
- ii) Retrieval of the page, assuming that it is the only page present in the bucket occurs at D .
- iii) The cost of finding an entry after scanning half the records on the page is $4RC$.
- iv) Fetching a record from the file is D . The total cost is,

$$H + D + 4RC + D \rightarrow (H + 2D + 4RC)$$

In case of many matched records the cost is,

$$H + D + 4RC + \text{one I/O for each record that qualifies.}$$

5) Range Selection Search: The cost of this is $B(D + RC)$.

Comparing Advantages and Disadvantages of Different File Organizations:

File Organization	Advantages	Disadvantages
1) Heap file	<ul style="list-style-type: none"> - Good storage efficiency - Rapid scanning - Insertion is fast 	<ul style="list-style-type: none"> - slow searches - slow deletion
2) Sorted file	<ul style="list-style-type: none"> - Good storage efficiency - Search is faster than heap file. 	<ul style="list-style-type: none"> - Insertion is slow. - Slow deletion.
3) Clustered file	<ul style="list-style-type: none"> - Good storage efficiency - Searches fast - Efficient insertion and deletion 	<ul style="list-style-type: none"> - Space overhead
4) Heap file with Un-clustered tree index.	<ul style="list-style-type: none"> - Fast insertion, deletion and searching 	<ul style="list-style-type: none"> - Scanning and range searches are slow.
5) Heap file with Un-clustered Hash index.	<ul style="list-style-type: none"> - Fast insertion, deletion and searching 	<ul style="list-style-type: none"> - Doesn't support range searches.

Dangling Pointer: Dangling pointer is a pointer that does not point to a valid object of the appropriate type. Dangling pointers arise when an object is deleted or de-allocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the de-allocated memory.

In object-oriented database, dangling pointer occur if we move or delete a record to which another record contains as pointer that pointer no longer points to the desired record.

Detecting Dangling pointer in object-oriented Databases: Mapping objects to files is similar to mapping tuples to files in a relational system, object data can be stored using file structures. Objects are identified by an object identifier (OID), the storage system needs a mechanism to locate given its OID.

Logical identifiers do not directly specify an objects physical location, must maintain an index that maps an OID to the object's actual location.

Physical identifiers encode the location of the object so the object can be found directly. Physical OIDs have the following parts.

- 1) A volume or file identifier.
- 2) A page identifier within the volume or file.

3) An offset within the page.

Physical OIDs may have a unique identifier. This identifier is stored in the object also and is used to detect reference via dangling pointers.

Vol. page offset	Unique_id
------------------	-----------

Unique_id	Data
-----------	------------

Indexes and Performance Tunning:

The performance of the system depends greatly on the indexes. This is done in terms of the expected work load.

Work Load Impact: Data entries that qualify particular selection criteria can be retrieved effectively by means of indexes. Two selection types are,

1) Equality 2) Range Selection.

1) Equality:

→ An equality query for a composite search key is defined as a search key in which each field is associated with a constant.

For **Example** data entries in a student file where rno = 15 and mrks = 90 can be retrieved by using equality query.

→ This is supported by Hash-file organization

2) Range Query: A range query for a composite search key is defined as a search key in which all the fields are not bounded to the constants.

Example: Data entries in a student file where rno = 15 with any mrks can be retrieved.

Thus, Tree-based indexing supports both the selection criteria as well as inserts, deletes and updates whereas only equality selection is supported by hash-based indexing apart from insertion, deletion and updation.

Advantages of using tree-structured indexes:

- 1) By using tree-structured indexes, insertion and deletion of data entries can be handled effectively.
- 2) It finds the correct leaf page faster than binary search in a sorted file.

Disadvantages:

The stored file pages are in accordance with the disk's order hence sequential retrieval of such pages is quicker which is not possible in tree-structured indexes.