

# New L J Institute of Engineering and Technology, Bodakdev

**Department: CSE (AIML) / IT**


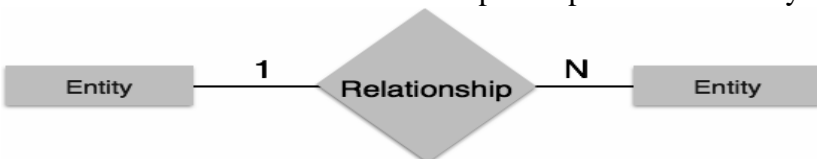

**Improvement Questions with Solution**

## **DATABASE MANAGEMENT SYSTEMS(3130703)**

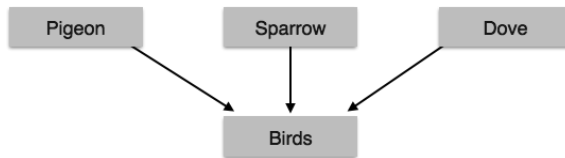
Sr. No	CHAPTER NO- 1: DATABASE SYSTEM ARCHITECTURE
1	<b>Explain the three level architecture of DBMS.</b>
Ans.	<div data-bbox="300 817 949 1176"><pre>graph TD; subgraph External_Level [External Level]; ES[External Schema]; end; subgraph Conceptual_Level [Conceptual Level]; CS[Conceptual Schema]; end; subgraph Internal_Level [Internal Level]; IS[Internal Schema]; end; DB[(Database)]; ES &lt;--&gt; External / Conceptual Mapping  CS; CS &lt;--&gt; Conceptual / Internal Mapping  IS; IS --- DB;</pre></div> <p>This figure shows three level database architecture.</p> <p>This framework is used to describe the structure of a specific database system. Here Mapping is used to transform the request and response between various database levels of architecture.</p> <p>In External / Conceptual mapping, it is necessary to transform the request from external level to conceptual schema.</p> <p>In Conceptual / Internal mapping, DBMS transform the request from the conceptual to internal level.</p> <p><b>1. Physical (Internal) Level:</b> This is the lowest level of abstraction which describes HOW data are actually stored in database. It describes physical storage of database. It is known as Internal Level also and shows complex low level data structure in detail.</p> <p><b>2. Logical (Conceptual) Level:</b> This is Next Higher level of abstraction which describes WHAT data to be stored in database. Also describe what kind of relationship is there among these data. It describe structure of whole database. Here information of internal details like implementation of structure is hidden. Programmers and DBA work at this level.</p>

	<p><b>3. View(External) Level:</b>  This is the highest level of abstraction which describe particular part of the database  End user interaction done at this level.  This level use simpler structure as many user do not want to all information, instead of that they want to access only some part of database. So this level simplifies their interaction with the system.  There are multiple types of views for same database.</p>
<b>2</b>	<b>What are types of database users? Explain various functions of Database Administrator?</b>
<b>Ans.</b>	<p>Those people who work with database can be categorized as database users and database administrator.</p> <p><b>Types of Database Users.</b></p> <ol style="list-style-type: none"> <li><b>1. Naïve Users:</b>  These are unsophisticated users who use system by invoking some application. Who don't have any DBMS knowledge but they frequently use the data base applications in their daily life to get the desired results. Clerks in any university is a naive user because they don't have any DBMS knowledge but they still use the database and perform their given task.</li> <li><b>2. Application Programmer:</b>  These are computer professional. They write application programs. They use different tools to build application or programs. These programs could be written in Programming languages such as Visual Basic, Developer, C, FORTRAN, COBOL etc.</li> <li><b>3. Sophisticated Users:</b>  This kind of user use system without writing programs. But they form their request using some database query language. Sophisticated users can be engineers, scientists, business analyst, who are familiar with the database.</li> <li><b>4. Specialized Users:</b>  This users are sophisticated users who writes specialized database application that do not fit in traditional database system</li> <li><b>5. Database Administrator:</b>  The database administrator (DBA) is a person who defines schema of database. The person who is having central control on database.</li> </ol> <p><b>Various functions of Database Administrator</b></p> <ol style="list-style-type: none"> <li><b>1. Schema Definition:</b> using DDL statements DBA creates original database schema.</li> <li><b>2. Schema and Physical organization modification:</b> It is the person who carry out changes in schema at the time of changing need of any organization.</li> <li><b>3. Granting and Authorization For data access:</b> It is DBA's responsibility to give different grant access to different types of database users. That will regulate which part of database can be accessed by which user. This authorization information must be kept in special system.</li> <li><b>4. Routine Maintenance:</b> DBA will do different task for maintenance like Periodically take backup of the database to prevent loss of data during failure</li> </ol>

	<p>Ensure that enough disk space is there to perform different operations. Monitor task running on system.</p>
<b>3</b>	<b>Explain Disadvantages of conventional file-based system compared to database management system.</b>
<b>Ans.</b>	<p><b>1. Data Redundancy:</b> As different programmers create files over long period and various files have different structure and different programming language. Same information may be duplicated in several place or files so this leads to data redundancy results in memory wastage. For an Example suppose some student has interest in music and maths both. Then details of that student (name, address, phone number) will be stored in file of music class as well as maths class.</p> <p><b>2. Data Inconsistency:</b> Because of data redundancy, it is possible that data may not be in consistent state. As we take same example of student then if student is changing his address or phone number then it is possible that changes will be reflected in one file like maths class but not in music class.</p> <p><b>3. Difficulty in Accessing Data:</b> Accessing data is not convenient and efficient in file processing system. Suppose bank clerk want data of those customers who live in particular postal code area. There is no such program written by designer. So to retrieve this data in file system clerk has two choice. Either extract needed information manually by generating list of all customers or ask programmer to write such program. And both methods are unsatisfactory. Because after some time clerk will need another information like this and same issue will be arise again.</p> <p><b>4. Limited Data Sharing:</b> Data are scattered in various files. Also different files may have different formats and these files may be stored in different folders may be of different departments. So, due to this data isolation, it is difficult to share data among different applications.</p> <p><b>5. Integrity Problems:</b> Data integrity means that the data contained in the database in both correct and consistent. For this purpose the data stored in database must satisfy correct and constraints. Suppose university is having data about salary of faculties. Now university wants that balance must not be less than zero. Application programmer will do necessary changes in code to enforce this constraints. But later on if we want to apply other one then it is difficult to apply changes in code each and every time.</p> <p><b>6. Atomicity Problems:</b> A computer or any device can fail any time. So in many application it is important that whenever system is having failure data should be restored in consistent state. Suppose we are having operation of transaction like 300 rupees transfer from one account to other then it is necessary that it should complete entirely or not at all. There is no middle state in transaction. Any operation on database must be atomic, this means, it must happen in it's entirely or not at all.</p> <p><b>7. Concurrent Access Anomalies:</b> Multiple users are allowed to access data simultaneously. This is for the sake of better performance and faster response. This can lead to inconsistent data as there is not concurrent access mechanism in conventional file system</p>

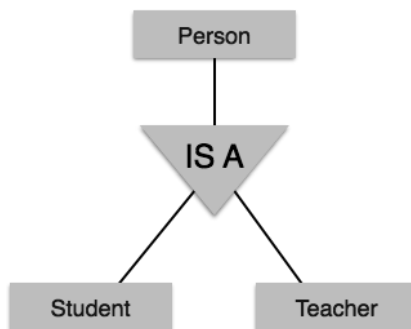
	<p><b>8. Security Problems:</b></p> <p>Not every user is intended to use or access all data. Some information are supposed to access by some users only. Database should be accessible to users in limited way. Each user should be allowed to access data concerning his requirements only. But to enforce such security constraints are difficult in file processing system.</p>
	<b>CHAPTER NO- 2: DATA MODELS</b>
<b>4</b>	<b>Explain mapping cardinalities and total participation with appropriate example.</b>
<b>Ans.</b>	<p>Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.</p> <p>Binary Relationship and Cardinality</p> <p>A relationship where two entities are participating is called a binary relationship. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.</p> <ul style="list-style-type: none"> <li> <p>One-to-one – When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.</p>  </li> <li> <p>One-to-many – When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.</p>  </li> <li> <p>Many-to-one – When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.</p>  </li> <li> <p>Many-to-many – The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.</p> </li> </ul>

	<div data-bbox="521 230 1197 385" data-label="Diagram"> <pre> graph LR     Entity1[Entity] --- N1[N] --- Relationship{Relationship}     Relationship --- N2[N] --- Entity2[Entity] </pre> </div> <p>Participation Constraints</p> <ul style="list-style-type: none"> <li>• Total Participation – Each entity is involved in the relationship. Total participation is represented by double lines.</li> <li>• Partial participation – Not all entities are involved in the relationship. Partial participation is represented by single lines.</li> </ul> <div data-bbox="446 656 1268 882" data-label="Diagram"> <pre> graph LR     Entity1[Entity] === Relationship{Relationship}     Relationship --- Entity2[Entity]     subgraph Labels         direction TB         L1[total participation] --&gt; Entity1         L2[partial participation] --&gt; Entity2     end </pre> </div>
5	<p><b>Define E-R diagram. Discuss generalization and specialization in E-R diagram with suitable diagram.</b></p>
Ans.	<p>The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases</p> <p>It has the power of expressing database entities in a conceptual hierarchical manner. As the hierarchy goes up, it generalizes the view of entities, and as we go deep in the hierarchy, it gives us the detail of every entity included.</p> <p>Going up in this structure is called generalization, where entities are clubbed together to represent a more generalized view. For example, a particular student named Mira can be generalized along with all the students. The entity shall be a student, and further, the student is a person. The reverse is called specialization where a person is a student, and that student is Mira.</p> <p><b>Generalization</b></p> <p>As mentioned above, the process of generalizing entities, where the generalized entities contain the properties of all the generalized entities, is called generalization. In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For example, pigeon, house sparrow, crow and dove can all be generalized as Birds.</p>



### Specialization

Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics. Take a group 'Person' for example. A person has name, date of birth, gender, etc. These properties are common in all persons, human beings. But in a company, persons can be identified as employee, employer, customer, or vendor, based on what role they play in the company.



Similarly, in a school database, persons can be specialized as teacher, student, or a staff, based on what role they play in school as entities.

**6 What is Integrity Constraint? Explain various types of Integrity constraints with Example.**

**Ans. Integrity Constraints**

The Set of rules which is used to maintain the quality of information are known as integrity constraints. Integrity constraints make sure about data intersection, update and so on. Integrity constraints can be understood as a guard against unintentional damage to the database.

Mainly Constraints on the relational database are of 4 types:

1. Domain constraints
2. Key constraints
3. Entity Integrity constraints
4. Referential integrity constraints

**1. Domain constraints :**

1. Every domain must contain atomic values(smallest indivisible units) it means composite and multi-valued attributes are not allowed.
2. We perform datatype check here, which means when we assign a data type to a column we limit the values that it can contain. Eg. If we assign the datatype of attribute age as int, we cant give it values other then int datatype.

**Example:**

EID	Name	Phone
01	Bikash Dutta	123456789 234456678

In the above relation, Name is a composite attribute and Phone is a multi-values attribute, so it is violating domain constraint.

## **2. Key Constraints or Uniqueness Constraints :**

1. These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.
2. A relation can have multiple keys or candidate keys(minimal superkey), out of which we choose one of the keys as primary key, we don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes.
3. Null values are not allowed in the primary key, hence Not Null constraint is also a part of key constraint.

**Example:**

EID	Name	Phone
01	Bikash	6000000009
02	Paul	9000090009
01	Tuhin	9234567892

In the above table, EID is the primary key, and first and the last tuple has the same value in EID ie 01, so it is violating the key constraint.

## **3. Entity Integrity Constraints :**

1. Entity Integrity constraints says that no primary key can take NULL value, since using primary key we identify each tuple uniquely in a relation.

**Example:**

EID	Name	Phone
01	Bikash	9000900099
02	Paul	600000009
NULL	Sony	9234567892

In the above relation, EID is made primary key, and the primary key can't take NULL values but in the third tuple, the primary key is null, so it is a violating Entity Integrity constraints.

#### 4. Referential Integrity Constraints :

1. The Referential integrity constraints is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.
2. This constraint is enforced through foreign key, when an attribute in the foreign key of relation R1 have the same domain(s) as the primary key of relation R2, then the foreign key of R1 is said to reference or refer to the primary key of relation R2.
3. The values of the foreign key in a tuple of relation R1 can either take the values of the primary key for some tuple in relation R2, or can take NULL values, but can't be empty.

#### Example:

EID	Name	DNO
01	Divine	12
02	Dino	22
04	Vivian	14

DNO	Place
12	Jaipur
13	Mumbai
14	Delhi

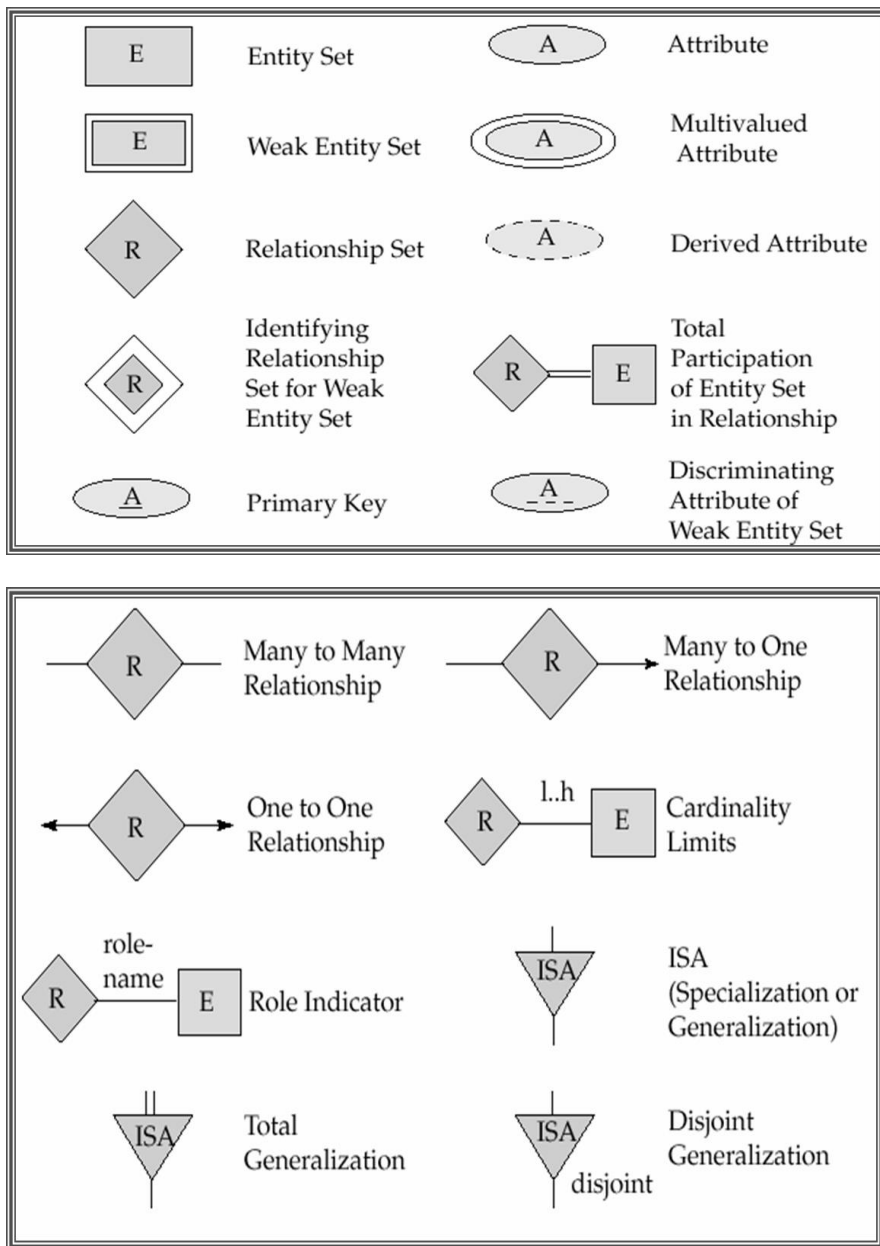
In the above, DNO of the first relation is the foreign key, and DNO in the second relation is the primary key. DNO = 22 in the foreign key of the first table is not allowed since DNO = 22 is not defined in the primary key of the second relation. Therefore, Referential integrity constraints is violated here.

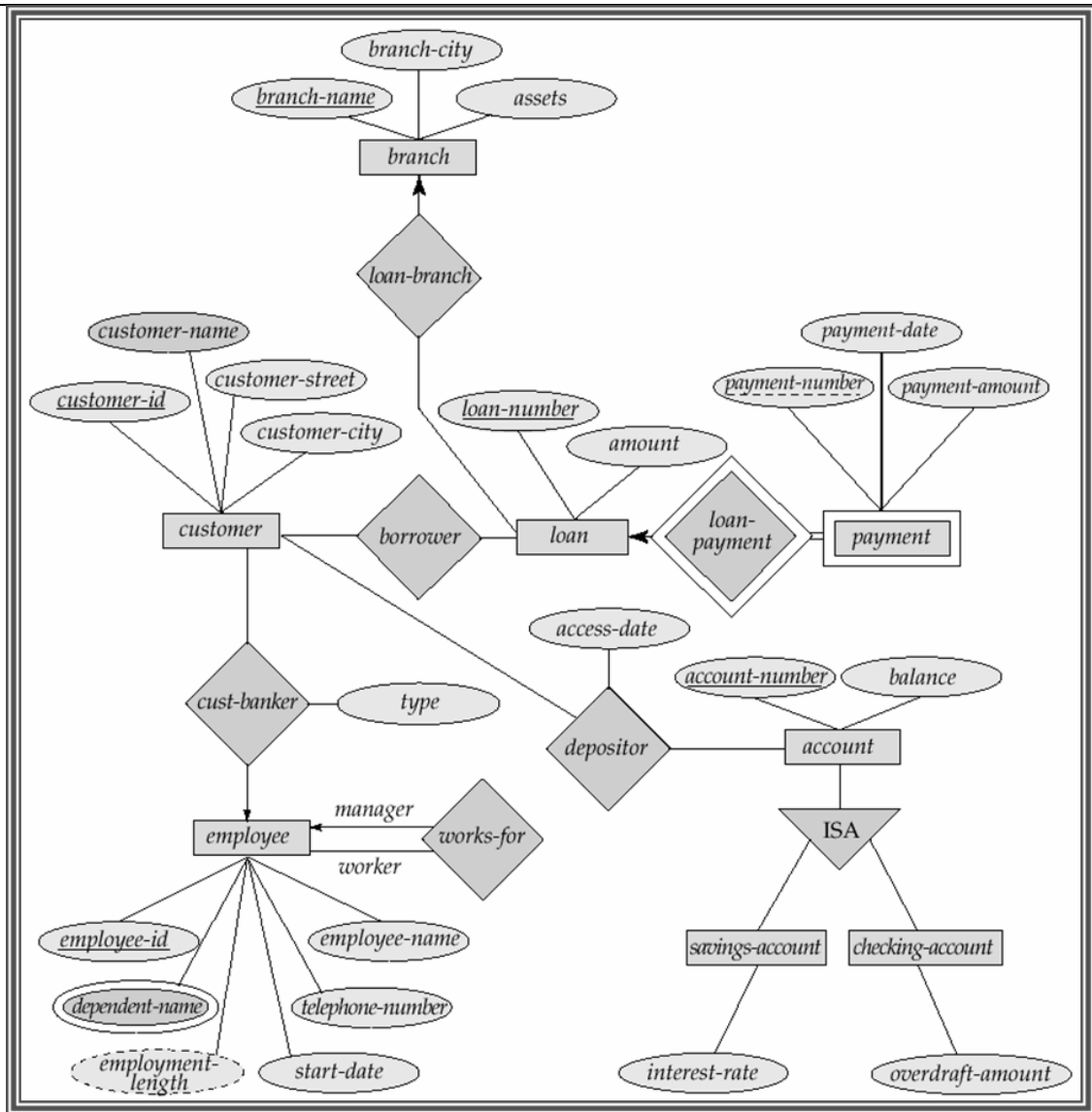


7

Which symbol used in E-R diagram? Draw an ER diagram for banking system.

Ans.





**8 Explain the difference between a weak and a strong entity set.**

**Ans. Strong Entity:**

A strong entity is not dependent on any other entity in the schema. A strong entity will always have a primary key. Strong entities are represented by a single rectangle. The relationship of two strong entities is represented by a single diamond.

Various strong entities, when combined together, create a strong entity set.

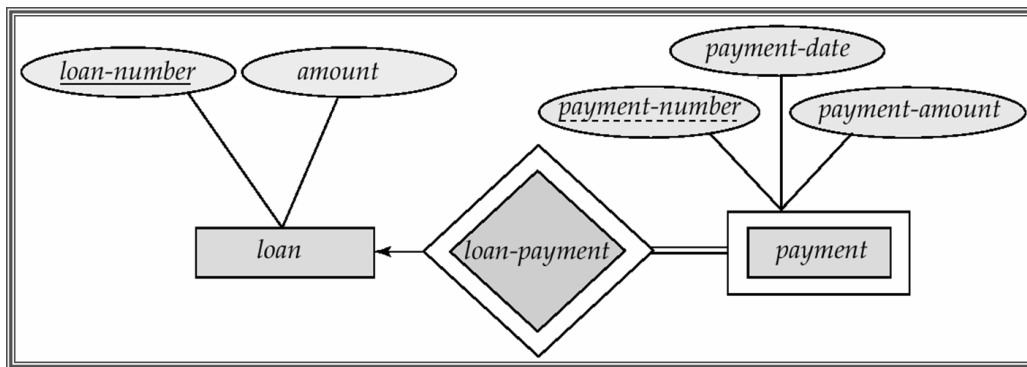
**Weak Entity:**

A weak entity is dependent on a strong entity to ensure its existence. Unlike a strong entity, a weak entity does not have any primary key. It instead has a partial discriminator key. A weak entity is represented by a double rectangle.

The relation between one strong and one weak entity is represented by a double diamond.

This relationship is also known as **identifying relationship**.

Consider a binary relationship below :



Since, we are not able to find the primary key for the payment-number payment entity set from its attributes. Attribute payment-number serves as a discriminator of the payment entity set and primary key for payment is (loan-number, payment-number).

### CHAPTER NO- 3: RELATIONAL QUERY LANGUAGES

**9 What is Relational Algebra? Enlist and Explain any three fundamental operation of Relational algebra.**

**Ans.** Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

The operations of relational algebra are as follows –

- Select
- Project
- Set operations
- Cartesian product
- Rename
- Join
- Division

**1. Select:**

It is used to fetch rows or tuples from table (relation).

We use the lowercase Greek letter sigma ( $\sigma$ ) to denote selection.

Syntax:

$$\sigma_{\text{predicate}}(\text{relation})$$

Example:

Consider Employee relation.

emp_id	emp_name	emp_city	salary
101	ram	ahmedabad	10000
102	shyam	surat	25000
103	ketan	rajkot	7000
104	mahesh	ahmedabad	15000

**Query:** List employee who have salary more than 10000.

$$\sigma_{\text{salary} > 10000}(\text{employee})$$

emp_id	emp_name	emp_city	salary
102	shyam	surat	25000
104	mahesh	ahmedabad	15000

We can also specify condition using and & or..

**Query:** List employee who have salary more than 10000 and belongs to ahmedabad.

$\sigma_{\text{salary} > 10000 \text{ and } \text{emp\_city} = \text{'ahmedabad'}}(\text{employee})$

## 2. Project:

It projects column(s) that satisfy a given predicate.

Syntax:

$\Pi_{A_1, A_2, \dots, A_n}(r)$

Where  $A_1, A_2, \dots, A_n$  are attribute names of relation  $r$ .

Duplicate rows are automatically eliminated, as relation is a set.

**Query:** List employee id and employee name only.

$\Pi_{\text{emp\_id}, \text{emp\_name}}(\text{employee})$

emp_id	emp_name
101	ram
102	shyam
103	ketan
104	mahesh

**Query:** List employee name who belongs to ahmedabad.

$\Pi_{\text{emp\_name}} \sigma_{\text{emp\_city} = \text{'ahmedabad'}}(\text{employee})$

emp_name
ram
mahesh

## 3. Cartesian Product:

The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.

It is denoted by  $\times$ .

Syntax:

$R_1 \times R_2$

Consider following relations:

	employee:	emp_dept				
	emp_id	emp_name	emp_city	salary	emp_id	emp_dept
	101	ram	ahmedabad	10000	101	admin
	102	shyam	surat	25000	102	account
					103	teaching
	employee × emp_dept					
	employee.emp_id	emp_name	emp_city	salary	emp_dept.emp_id	Emp_dept
	101	ram	ahmedabad	10000	101	admin
	101	ram	ahmedabad	10000	102	account
	101	ram	ahmedabad	10000	103	teaching
102	shyam	surat	25000	101	admin	
102	shyam	surat	25000	102	account	
102	shyam	surat	25000	103	teaching	
<b>Query:</b> List information of employee who have allocated department.						
$\sigma_{employee.emp\_id=emp\_dept.emp\_id}(employee \times emp\_dept)$						
	employee.emp_id	emp_name	emp_city	salary	emp_dept.emp_id	emp_dept
	101	ram	ahmedabad	10000	101	admin
	102	shyam	surat	25000	102	account
10	<b>Explain different types of outer join with examples.</b>					
Ans.	An outer join is basically of three types:  1. Left outer join 2. Right outer join 3. Full outer join Consider following relations: employee					
	emp_id	emp_name	emp_city	salary	emp_dept	
	emp_id	emp_dept				
	101	ram	ahmedabad	10000	101	admin
	102	shyam	surat	25000	102	account
	103	manish	ahmedabad	12000	104	teaching
<b>1. Left Outer Join:</b>						
	• In this, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.					

- It is denoted by  $\bowtie$ .



**Example:** employee  $\bowtie$  emp\_dept

emp_id	emp_name	emp_city	salary	emp_dept
101	ram	ahmedabad	10000	admin
102	shyam	surat	25000	account
103	manish	ahmedabad	12000	NULL

## 2. Right Outer Join

- In this, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.
- It is denoted by  $\bowtie$ .



**Example:** employee  $\bowtie$  emp\_dept

emp_id	emp_name	emp_city	salary	emp_dept
101	ram	ahmedabad	10000	admin
102	shyam	surat	25000	account
104	NULL	NULL	NULL	teaching

## 3. Full Outer Join

- In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.
- It is denoted by  $\bowtie$ .

**Example:** employee  $\bowtie$  emp\_dept

emp_id	emp_name	emp_city	salary	Emp_dept
101	ram	ahmedabad	10000	admin
102	shyam	surat	25000	Account
103	manish	ahmedabad	12000	NULL
104	NULL	NULL	NULL	teaching

CHAPTER NO- 4: RELATIONAL DATABASE DESIGN																
11	<b>What is Functional dependency? Explain all its types with example.</b>															
Ans.	<p>Functional Dependency (FD) provides constraint between various columns (attributes) in Table (Relation).</p> <p>In R(X,Y), X and Y represent attributes where Y is functionally dependent on other column X, or we can also state that X functionally determines Y. This dependency can be denoted by ( <math>\longrightarrow</math> )</p> <table><tr><th>RollNo</th><th>Name</th><th>Marks</th><th>Dept</th><th>Course</th></tr><tr><td>1</td><td>Astha</td><td>9</td><td>CE</td><td>c1</td></tr><tr><td>2</td><td>Parth</td><td>8</td><td>IT</td><td>c1</td></tr></table> <p>RollNo <math>\rightarrow</math> Name RollNo <math>\rightarrow</math> Name, Marks</p> <p><b><u>Types of Functional Dependencies</u></b></p> <ol style="list-style-type: none"><li><b>Full Functional Dependency</b><ul style="list-style-type: none"><li>In a relation, the attribute B is fully functional dependent on A if <b>B is functionally dependent on A, but not on any proper subset of A.</b></li><li>Eg. {RollNo, Sem, Dept} <math>\rightarrow</math> SPI</li><li>We need all three {RollNo, Sem, Dept} to find SPI.</li></ul></li><li><b>Partial Functional Dependency</b><ul style="list-style-type: none"><li>In a relation, the attribute B is partial functional dependent on A if <b>B is functionally dependent on A as well as on any proper subset of A.</b></li><li>If there is some attribute that can be removed from A and the still dependency holds then it is partial functional dependency.</li><li>Eg. {EnrollmentNo, Dept} <math>\rightarrow</math> SPI</li><li><b>Enrollment_No is sufficient to find SPI</b>, Department Name is not required to find SPI.</li></ul></li><li><b>Transitive Functional Dependency</b><ul style="list-style-type: none"><li>In a relation, if attribute(s) <b>A <math>\rightarrow</math> B</b> and <b>B <math>\rightarrow</math> C</b>, then <b>C is transitively depends on A via B.</b></li><li>Eg. Staff_No <math>\rightarrow</math> Branch_No ; Branch_No <math>\rightarrow</math> Branch_Address then Staff_No <math>\rightarrow</math> Branch_Address</li></ul></li><li><b>Trivial Functional Dependency</b><ul style="list-style-type: none"><li><b>X <math>\rightarrow</math> Y</b> is trivial FD if <b>Y is a subset of X</b></li><li>Eg. {Roll_No, Department_Name} <math>\rightarrow</math> Roll_No</li></ul></li><li><b>Nontrivial Functional Dependency</b><ul style="list-style-type: none"><li><b>X <math>\rightarrow</math> Y</b> is nontrivial FD if <b>Y is not a subset of X</b></li><li>Eg. {Roll_No, Department_Name} <math>\rightarrow</math> Student_Name</li></ul></li></ol>	RollNo	Name	Marks	Dept	Course	1	Astha	9	CE	c1	2	Parth	8	IT	c1
RollNo	Name	Marks	Dept	Course												
1	Astha	9	CE	c1												
2	Parth	8	IT	c1												
12	<b>Given R= (A, B, C, G, H, I). The following set F of functional dependencies holds</b> <b>A <math>\rightarrow</math> B   A<math>\rightarrow</math>C   CG <math>\rightarrow</math>H   CG <math>\rightarrow</math>I   B <math>\rightarrow</math>H</b> Compute AG <sup>+</sup> . Is AG a candidate key?															
Ans.	<p>We have R= (A, B, C, G, H, I). and F={ A <math>\rightarrow</math> B   A<math>\rightarrow</math>C   CG <math>\rightarrow</math>H   CG <math>\rightarrow</math>I   B <math>\rightarrow</math>H }</p> <p>1. Create an empty set closure of AG<sup>+</sup> = { } and add all the attributes that are functionally determined by AG.</p>															

	<ol style="list-style-type: none"> <li>2. As <math>AG \rightarrow AG</math> (self-determination rule) we add AG to the empty set to get <math>AG^+ = \{AG\}</math></li> <li>3. Then we have <math>A \rightarrow B, A \rightarrow C</math> so we can add B and C to the set to get <math>AG^+ = \{ABCG\}</math>. We choose to consider functional dependencies of both A as well as G because both are subsets of Set S.</li> <li>4. Now the set contains <math>AG^+ = \{ABCG\}</math>, we consider the subsets of the set elements and check their functional dependencies :</li> <li>5. One such subset is CG (As <math>CG \subset ABCG</math>) and the functional dependencies are <math>CG \rightarrow H</math> and <math>CG \rightarrow I</math>.</li> <li>6. Thus, we can add H and I to the set to get <math>AG^+ = \{ABCGHI\}</math>.</li> <li>7. As all elements are covered in <math>AG^+</math> so we can stop.</li> </ol> <p><b>Here <math>AG^+ = \{ABCGHI\}</math> contains all attributes from relation <math>R(A, B, C, G, H, I)</math>. So we can say that it is a candidate key.</b></p>
<b>13</b>	<b>Explain Armstrong's Axioms in detail.</b>
<b>Ans.</b>	<p>Armstrong's axioms are a <b>set of rules used to infer (derive) all the functional dependencies</b> on a relational database.</p> <ol style="list-style-type: none"> <li>1. Reflexivity <ul style="list-style-type: none"> <li>– If <b>B is a subset of A</b></li> <li>then <b><math>A \rightarrow B</math></b></li> </ul> </li> <li>2. Augmentation <ul style="list-style-type: none"> <li>– If <b><math>A \rightarrow B</math></b></li> <li>then <b><math>AC \rightarrow BC</math></b></li> </ul> </li> <li>3. Transitivity <ul style="list-style-type: none"> <li>– If <b><math>A \rightarrow B</math> and <math>B \rightarrow C</math></b></li> <li>– then <b><math>A \rightarrow C</math></b></li> </ul> </li> <li>4. Pseudo Transitivity <ul style="list-style-type: none"> <li>– If <b><math>A \rightarrow B</math> and <math>BD \rightarrow C</math></b></li> <li>– then <b><math>AD \rightarrow C</math></b></li> </ul> </li> <li>5. Self-determination <ul style="list-style-type: none"> <li>– <b><math>A \rightarrow A</math></b></li> </ul> </li> <li>6. Decomposition <ul style="list-style-type: none"> <li>– If <b><math>A \rightarrow BC</math></b></li> <li>then <b><math>A \rightarrow B</math> and <math>A \rightarrow C</math></b></li> </ul> </li> <li>7. Union <ul style="list-style-type: none"> <li>– If <b><math>A \rightarrow B</math> and <math>A \rightarrow C</math></b></li> <li>– then <b><math>A \rightarrow BC</math></b></li> </ul> </li> <li>8. Composition <ul style="list-style-type: none"> <li>– If <b><math>A \rightarrow B</math> and <math>C \rightarrow D</math></b></li> <li>– then <b><math>AC \rightarrow BD</math></b></li> </ul> </li> </ol>



**14      What is Normalization? Explain 1NF, 2NF, 3NF, BCNF with example.**

**Ans.** Normalization is the process of organizing the data in the database.

Need:-

To Remove Redundant data

To Minimize disk space

To Reduce chance of data errors

To increase performance

To improve consistency

Type:

1NF, 2NF, 3NF, BCNF, 4NF, 5NF

**First Normal Form(1NF):**

It contains atomic value. Which means attribute must hold single value

Values stored in column should be same domain.

Example:

EMP_ID	EMP_PHONE
1	2826385, 0647382
2	4783832
3	0372389, 9830302

Here in this table we are having 2 column where EMP\_PHONE is having multiple value for single EMP\_ID. So this is not in first normal form.

We can make it in 1NF. The conversion is as follows.

EMP_ID	EMP_PHONE
1	2826385
1	0647382
2	4783832
3	0372389
3	9830302

## Second Normal Form:

To be in second normal form

1. Table must be in first normal form
2. It should **not have** partial functional dependency

Faculty		
FID	SUBJECT	F_AGE
1	Chemistry	35
2	Biology	40
3	English	29

In this table if we have  $FD = \{FID, SUBJECT\} \rightarrow F\_AGE$  then here  $F\_AGE$  is dependent on  $FID$  and  $SUBJECT$  as per  $FD$ . But here we are having partial dependency.

As to determine faculty Age we do not require subject. Just  $FID$  is enough. So in actual  $F\_AGE$  is dependent on  $FID$  only.

So to remove this dependency we need to decompose this table.

So after decomposing we will have 2 tables.

FacultySubject	
FID	SUBJECT
1	Chemistry
2	Biology
3	English

FacultyAge	
FID	F_AGE
1	35
2	40
3	29

**Third Normal Form:**

To be in third normal form

1. Table must be in second normal form
2. It should not have Transitive functional dependency

Example:

S_ID	S_NAME	S_ZIP	S_CITY
2	Harry	201010	Noida
3	Stephan	02228	Boston

Here we have dependencies like

$S\_ID \rightarrow S\_ZIP$

$S\_ZIP \rightarrow S\_CITY$

So using transitivity rule we are getting

$S\_ID \rightarrow S\_CITY$

So this table is not in third normal form.

To convert this table we need to decompose this table into 2 tables

Student_Detail		
S_ID	S_NAME	S_ZIP
2	Harry	201010
3	Stephan	02228

ZipCodes	
S_ZIP	S_CITY
201010	Noida
02228	Boston

### Boyce Codd Normal Form(BCNF)

Relation is in BCNF if every functional dependency  $X \rightarrow Y$ , X is the super key of the table.

The Relation should be in 3NF, and for every FD, LHS is super key.

Consider last example of 3NF:

- S\_ID is the super-key in the relation Student\_Detail and S\_Zip is the super-key in the relation ZipCodes. So,  
 $S\_ID \rightarrow S\_Name, S\_Zip$

and

$$S\_Zip \rightarrow S\_City$$

Which confirms that both the relations are in BCNF.

**15 What is decomposition? Why is it required? Explain the difference between lossy decomposition and non-loss decomposition with example**

**Ans. Decomposition**

- When relation or table is not in appropriate form at that time we need to do decomposition.
- It is the process where we break table into multiple tables.
- If we do not go with proper decomposition then it can lead loss of information

#### Lossless Decomposition

If we decompose our relation R into R1 and R2 and then we join this R1 and R2 and getting same relation R (on natural join). Then it is known as lossless.

Example:

#### Employee Table:

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
1	A	29	INDIA	D1	Operations
2	B	32	INDIA	D2	HR
3	C	22	US	D3	Finance

Here we have Employee table. Suppose we are decomposing this table into 2 tables.

EmployeeDetails			
Emp_ID	Emp_Name	Emp_Age	Emp_Location
1	A	29	INDIA
2	B	32	INDIA
3	C	22	US

EmployeeDepartment		
Dept_ID	Emp_ID	Dept_Name
D1	1	Operations
D2	2	HR
D3	3	Finance

Here we decomposed original table into two different table and there is one common attribute in this two table which is Emp\_ID.

So if we perform natural join on these two tables then we will get same relation Employee like original.

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
1	A	29	INDIA	D1	Operations
2	B	32	INDIA	D2	HR
3	C	22	US	D3	Finance

**So here as we are getting same relation like before. We can say that this is Lossless or Non-Loss Decomposition**

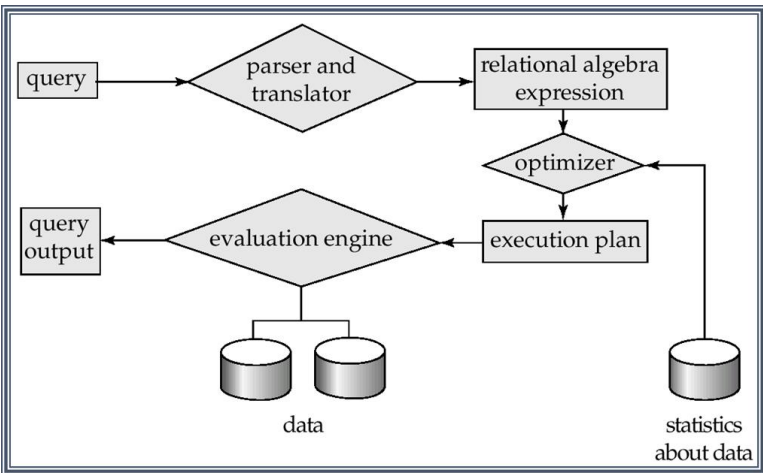
### **Lossy Decomposition**

When a relation is decomposed into two or more relational schemas, the loss of information is unavoidable when the original relation is retrieved.

So if we take same example of Employee table and now if we divide this table like below

EmployeeDetails			
Emp_ID	Emp_Name	Emp_Age	Emp_Location
1	A	29	INDIA
2	B	32	INDIA
3	C	22	US

EmployeeDepartment	
Dept_ID	Dept_Name
D1	Operations
D2	HR
D3	Finance

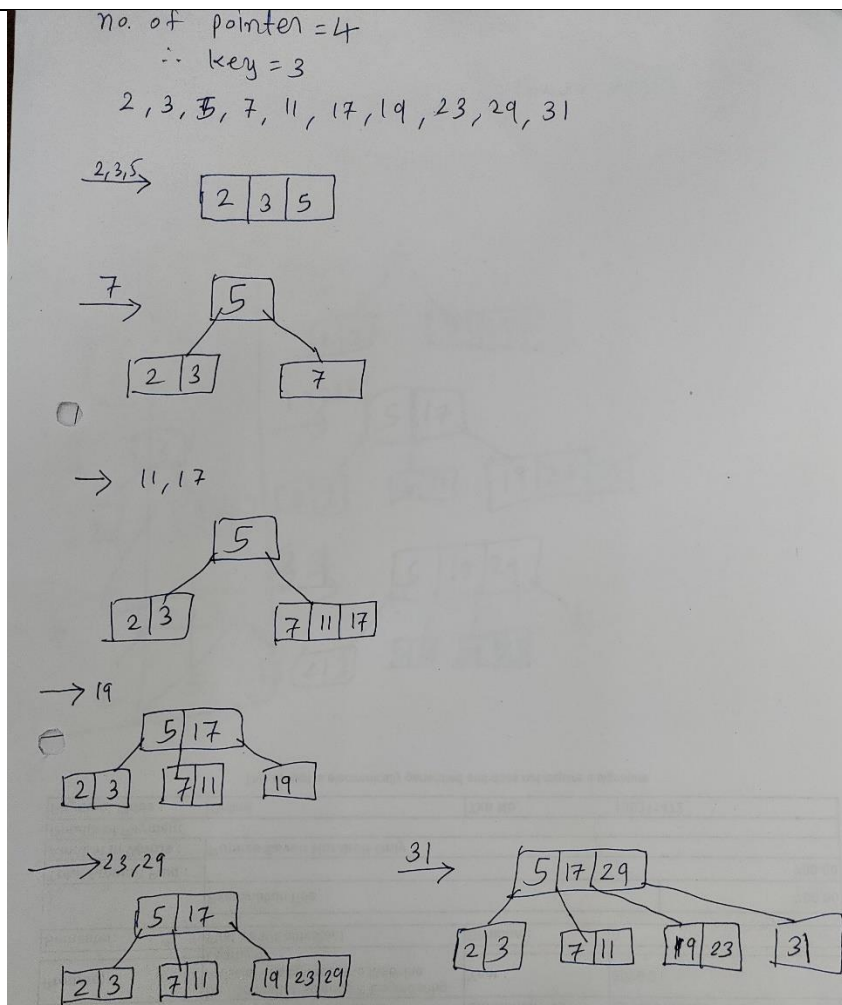
	<p>Here in second table we are just taking Dept_ID and Dept_Name. So there is no common attribute between these tables.</p> <p>So in this case if we join this two tables then we will get unwanted records which are not useful and not like the original relation Employee. So this decomposition is known as Lossy Decomposition.</p>
	<b>CHAPTER NO- 5: QUERY PROCESSING AND OPTIMIZATION</b>
<b>16</b>	<b>Discuss various steps of query processing with a diagram.</b>
<b>Ans.</b>	<p><b>Query Processing</b> is a translation of high-level queries into low-level expression. It is a step wise process that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result.</p> <p>It requires the basic concepts of relational algebra and file structure. It refers to the range of activities that are involved in extracting data from the database. It includes translation of queries in high-level database languages into expressions that can be implemented at the physical level of the file system. The activities involved in parsing, validating, execution and optimizing a query is called Query Processing.</p>  <pre> graph TD     query[query] --&gt; parser{parser and translator}     parser --&gt; algebra[relational algebra expression]     algebra --&gt; optimizer{optimizer}     optimizer --&gt; plan[execution plan]     plan --&gt; engine{evaluation engine}     engine --&gt; output[query output]     engine --- data[(data)]     engine --- stats[(statistics about data)]   </pre> <p><b>Step 1</b></p> <p><b>Parsing and translation</b></p> <p>While parsing, the database performs the checks like, Syntax check after converting the query into relational algebra.</p> <p><b>Step 2</b></p> <p><b>Optimizer</b> – In this stage, the database has to perform a hard parse at least for one unique DML statement and it has to do optimization during this parse. This database never optimizes DDL unless it includes a DML component.</p> <p>It is a process where multiple query execution plans for satisfying a query are examined and the most efficient query plan is satisfied for execution.</p> <p>Database catalogue stores the execution plans and then optimizer passes the lowest cost plan for execution.</p> <p><b>Step 3</b></p> <p><b>Execution Engine</b> – Execution engine is helpful to run the query and display the</p>

	required result.
<b>17</b>	<b>What is Optimization? Give methods of Query Optimization</b>
<b>Ans.</b>	<p>There are alternative ways of evaluating a given query as:</p> <ul style="list-style-type: none"> <li>•Equivalent expressions</li> <li>•Different algorithms for each operation</li> <li>•Cost difference evaluating a query</li> </ul> <p>We need to estimate the cost of operations. It depends critically on statistical information about relations which the database must maintain.</p> <ul style="list-style-type: none"> <li>•E.g. number of tuples, number of distinct values for join attributes, etc. We also have to estimate statistics for intermediate results to compute the cost of complex expressions.</li> </ul> <p>We have two major types of query estimation:</p> <ol style="list-style-type: none"> <li>1. Cost Based Estimation</li> <li>2. Heuristic Optimization</li> </ol> <p><b>Cost Based Estimation</b></p> <p>Cost is generally measured as total elapsed time for answering a query. Many factors contribute to time cost: <i>disk accesses</i>, <i>CPU</i>, or even network <i>communication</i>. Typically disk access is the predominant cost, and is also relatively easy to estimate. Measured by taking into account</p> <p>Number of seeks * average-seek-cost</p> <p>Number of blocks read * average-block-read-cost</p> <p>Number of blocks written * average-block-write-cost</p> <p>Cost to write a block is greater than cost to read a block data is read back after being written to ensure that the write was successful.</p> <p><b>Heuristic Optimization</b></p> <p>Cost-based optimization is expensive, even with dynamic programming. Systems may use <i>heuristics</i> to reduce the number of choices that must be made in a cost-based fashion.</p> <p>Heuristic optimization transforms the query-tree by using a set of rules that typically (but not in all cases) improve execution performance:</p> <ul style="list-style-type: none"> <li>•Perform selection early (reduces the number of tuples)</li> <li>•Perform projection early (reduces the number of attributes)</li> <li>•Perform most restrictive selection and join operations before other similar operations.</li> </ul> <p>Some systems use only heuristics, others combine heuristics with partial cost-based optimization.</p>

## CHAPTER NO- 6 : STORAGE STRATEGIES

- 18** Construct a B-tree for the following set of key values:  
(2,3,5,7,11,17,19,23,29,31)  
Assume that the tree is initially empty and values are added in ascending order.  
Consider the number of pointers in each node as four

**Ans.**



- 19** Explain Dense and Sparse indices in detail.

**Ans.** Dense Index:

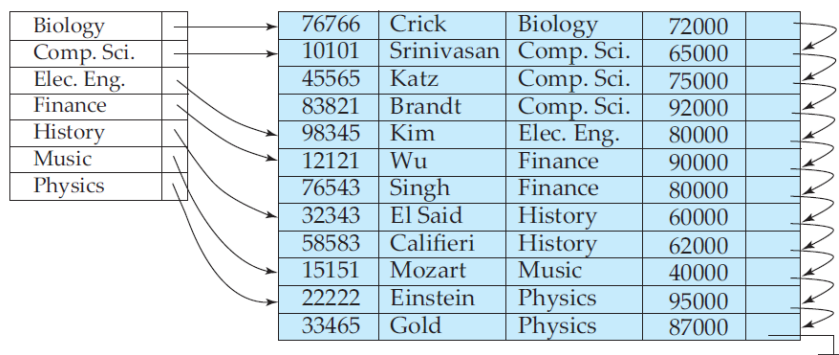
- In dense index, there is an index record for every search key value in the database.
- This makes searching faster but requires more space to store index records itself.
- Index records contain search key value and a pointer to the actual record on the disk.

Example: consider this relation



ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

If we apply index on department then

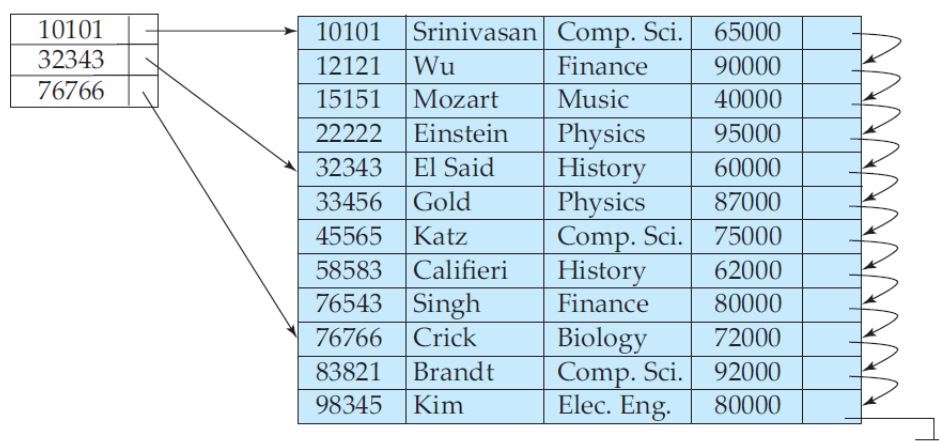


### Sparse Index:

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

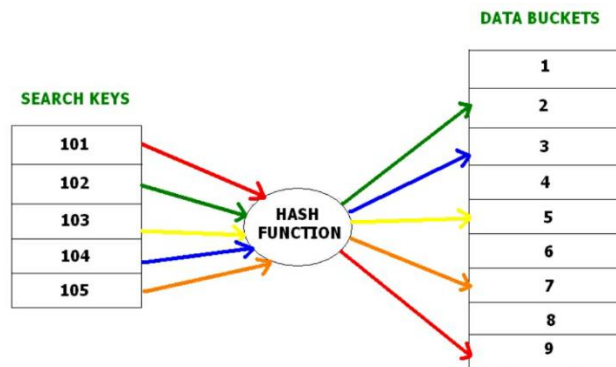
Example:

We created index on ID



**20 Explain the use of Hashing to store data.**

- Ans.**
- Hashing is an efficient technique to directly search the location of desired data on the disk without using index structure.
  - Data is stored at the data blocks whose address is generated by using hash function.
  - The memory location where these records are stored is called as data block or data bucket.



Example:

Consider hash table as below:

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

**Keys to be inserted:**

33, 45, 56, 12, 19

**Hash Function:**

Key Modulo (%) 10

Index	Key
0	
1	
2	12
3	33
4	
5	45
6	56
7	
8	
9	19

### Types of Hash Function

There are so many types of hashing functions, frequently used are:

- Division Method (Modulo)
- Multiplication Method
- Mid-Square Method
- Folding Method

#### **Multiplication Method:**

$$H(k) = \text{floor}(m (kA \bmod 1))$$

where  $(kA \bmod 1)$  gives the fractional part of  $kA$   
and  $m$  is the total number of indices in the hash table

Let  $A = 0.618033$ ,  $m = 1000$ ,  $k = 12345$

$$H(k) = \text{floor}(1000 (12345 \times 0.618033 \bmod 1))$$

$$H(k) = \text{floor}(1000 (7629.617385 \bmod 1))$$

$$H(k) = \text{floor}(1000 (0.617385))$$

$$H(k) = \text{floor}(617.385)$$

$$H(k) = 617$$

#### **Mid-square Method:**

1234

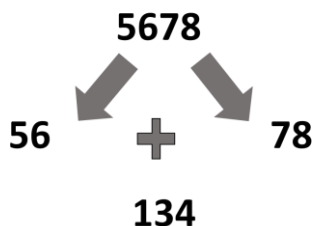
$(1234)^2$

=1522756



27

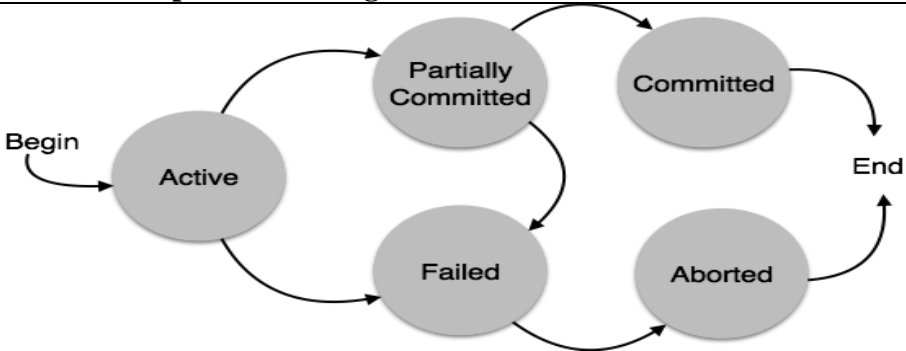
#### **Folding Method**



## **CHAPTER NO- 7 : TRANSACTION PROCESSING**

**21 List and explain ACID properties with respect to Database transaction**

**Ans.** A transaction is a very small logical unit of a program and it may contain several low level 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.

	<p><b>Atomicity</b> – 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.</p> <p><b>Consistency</b> – 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.</p> <p><b>Durability</b> – 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.</p> <p><b>Isolation</b> – 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.</p>
22	<b>Draw and explain state diagram of transaction.</b>
Ans.	 <pre> graph LR     Begin((Begin)) --&gt; Active((Active))     Active --&gt; PartiallyCommitted((Partially Committed))     Active --&gt; Failed((Failed))     PartiallyCommitted --&gt; Committed((Committed))     PartiallyCommitted --&gt; Failed     Committed --&gt; End((End))     Failed --&gt; Aborted((Aborted))     Aborted --&gt; End </pre> <p>A transaction in a database can be in one of the following states –</p> <ul style="list-style-type: none"> <li>● <b>Active</b> – In this state, the transaction is being executed. This is the initial state of every transaction.</li> <li>● <b>Partially Committed</b> – When a transaction executes its final operation, it is said to be in a partially committed state.</li> <li>● <b>Failed</b> – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.</li> <li>● <b>Aborted</b> – If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts –             <ul style="list-style-type: none"> <li>○ Re-start the transaction</li> <li>○ Kill the transaction</li> </ul> </li> <li>● <b>Committed</b> – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.</li> </ul>

23

**Explain conflict serializability and view serializability.**

Ans.

Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

If a schedule  $S$  can be transformed into a schedule  $S'$  by a series of swaps of non-conflicting instructions, we say that  $S$  and  $S'$  are conflict equivalent. We say that a schedule  $S$  is conflict serializable if it is conflict equivalent to a serial schedule.

**Conflict Equivalence**

Instructions  $l_i$  and  $l_j$  of transactions  $T_i$  and  $T_j$  respectively, conflict if and only if there exists some item  $Q$  accessed by both  $l_i$  and  $l_j$ , and at least one of these instructions wrote  $Q$ .

1.  $l_i = \text{read}(Q)$ ,  $l_j = \text{read}(Q)$ .  $l_i$  and  $l_j$  don't conflict.
2.  $l_i = \text{read}(Q)$ ,  $l_j = \text{write}(Q)$ . They conflict.
3.  $l_i = \text{write}(Q)$ ,  $l_j = \text{read}(Q)$ . They conflict
4.  $l_i = \text{write}(Q)$ ,  $l_j = \text{write}(Q)$ . They conflict

For example:

Schedule A below can be transformed into Schedule B, a serial schedule where  $T_2$  follows  $T_1$ , by series of swaps of non-conflicting instructions. Therefore Schedule A is conflict serializable.

**Schedule A**

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

**Schedule B**

T1	T2
R(A)	
W(A)	
R(B)	
W(B)	

	R(A)
	W(A)
	R(B)
	W(B)

#### View serializability

Let  $S$  and  $S'$  be two schedules with the same set of transactions.  $S$  and  $S'$  are view equivalent if the following three conditions are met:

1. For each data item  $Q$ , if transaction  $T_i$  reads the initial value of  $Q$  in schedule  $S$ , then transaction  $T_i$  must, in schedule  $S'$ , also read the initial value of  $Q$ .
2. For each data item  $Q$  if transaction  $T_i$  executes  $\text{read}(Q)$  in schedule  $S$ , and that value was produced by transaction  $T_j$  (if any), then transaction  $T_i$  must in schedule  $S'$  also read the value of  $Q$  that was produced by transaction  $T_j$ .
3. For each data item  $Q$ , the transaction (if any) that performs the final  $\text{write}(Q)$  operation in schedule  $S$  must perform the final  $\text{write}(Q)$  operation in schedule  $S'$ .

Schedule  $S$  is view serializable, if it is view equivalent to a serial schedule. Every conflict serializable schedule is also view serializable.

•Example— a schedule which is view-serializable but not conflict serializable.

T1	T2	T3
R(Q)		
	W(Q)	
W(Q)		
		W(Q)

Every view serializable schedule that is not conflict serializable has blind writes.

#### 24 What is locking? Explain Two phase locking and its types.

**Ans.** Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Data items can be locked in two modes :

1. *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.
2. *shared (S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction.

Lock requests are made to the concurrency-control manager. Transactions can proceed only after a request is granted. A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions.

	<p>Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item. If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.</p> <p>The Two-Phase Locking Protocol</p> <p>This is a protocol which ensures conflict-serializable schedules.</p> <p>Phase 1: Growing Phase</p> <ul style="list-style-type: none"> <li>- transaction may obtain locks</li> <li>- transaction may not release locks</li> </ul> <p>Phase 2: Shrinking Phase</p> <ul style="list-style-type: none"> <li>- transaction may release locks</li> <li>- transaction may not obtain locks</li> </ul> <p>The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their lock points (i.e. the point where a transaction acquired its final lock). Two-phase locking does not ensure freedom from deadlocks. Cascading roll-back is possible under two-phase locking. To avoid this, follow a modified protocol called <b>strict two-phase locking</b>. Here a transaction must hold all its exclusive locks till it commits/aborts.</p> <p><b>Rigorous two-phase locking</b> is even stricter: here all locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit.</p>
25	<b>Explain log based recovery and mention both its types.</b>
Ans.	<p>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.</p> <p>Log-based recovery works as follows –</p> <ul style="list-style-type: none"> <li>• When a transaction enters the system and starts execution, it writes a log about it.</li> <li>• When transaction <math>T_i</math> starts, it registers itself by writing a <math>\langle T_i \text{ start} \rangle</math> log record</li> <li>• Before <math>T_i</math> executes <b>write</b>(<math>X</math>), a log record <math>\langle T_i, X, V_1, V_2 \rangle</math> is written, where <math>V_1</math> is the value of <math>X</math> before the write (the <b>old value</b>), and <math>V_2</math> is the value to be written to <math>X</math> (the <b>new value</b>).</li> <li>• When <math>T_i</math> finishes its last statement, the log record <math>\langle T_i \text{ commit} \rangle</math> is written.</li> <li>• Two approaches using logs <ol style="list-style-type: none"> <li>1. Deferred database modification</li> <li>2. Immediate database modification</li> </ol> </li> </ul>

### Deferred database modification

Database not updated immediately. Only the log file is updated on each transaction. When a transaction reaches commit, the database is physically updated from the log file. If a transaction fails before reaching to commit, then it will not have changed the database anyway. **Hence, no need for UNDO operation. The REDO operation is required to record the operation from the log files to the physical file.** So this technique is called as **NO UNDO/REDO algorithm**.

Example:

T1	T2
Read(A,a) a=a-10 Write(A,a) Read(B,b) b=b+10 Write(B,b)	Read(C,c) c=c-20 write(C,c)

If T1 And T2 Execute serially, initially A=100, B=200, C=300

Log Records:

T1	Database
<b>&lt;T<sub>1</sub> start&gt;</b> <b>&lt;T<sub>1</sub> ,A, 90&gt;</b> <b>&lt;T<sub>1</sub> ,B, 210&gt;</b>	A=90 B=210
<b>&lt;T<sub>2</sub> start&gt;</b> <b>&lt;T<sub>2</sub> ,C, 280&gt;</b>	C=280

### Crash of database:

- Just after write(B,b): T1 deleted from log as commit of T1 is not executed.
- Just after write(C,c): redo(T1) is done as <t1.commit> get executed.so A=90, B=210, C=300
- Just after <T2,Commit> : redo(T1) and Redo(T2) execute. so A=90, B=210, C=280



**Immediate database modification:**

It modifies the database after a write operation, database modification is immediately done when a transaction performs an update/ write operation. Update log records maintain both old and new values of data items. The recovery system uses two operations, which are as follows –

Undo( $T_i$ ) – All data items updated by the transaction  $T_i$ , are set to old value.

Redo( $T_i$ ) – All data items updated by the transaction  $T_i$  are set to a new value.

Log Records (For above example):

T1	database
$\langle T_1 \text{ start} \rangle$	
$\langle T_1, A, 90 \rangle$	A=90
$\langle T_1, B, 210 \rangle$	B=210
$\langle T_2 \text{ start} \rangle$	
$\langle T_2, C, 280 \rangle$	C=280

Crash of database:

- Just after write(B,b): UNDO( $T_1$ ) is executed.
- Just after write(C,c): both REDO and UNDO will be executed. UNDO( $T_2$ ) and REDO( $T_1$ ), so A=90, B=210, C=300
- Just after  $\langle T_2, \text{Commit} \rangle$  : redo( $T_1$ ) and Redo( $T_2$ ) execute. so A=90, B=210, C=280

**26 Explain deadlock prevention, deadlock detection and recovery.**

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

Consider the following two transactions:

$T_1$ :	write (A)	$T_2$ :	write(B)
	write(B)		write(A)

## Schedule with deadlock

Schedule with deadlock

$T_1$	$T_2$
lock-X on $A$ write ( $A$ )      wait for lock-X on $B$	lock-X on $B$ write ( $B$ ) wait for lock-X on $A$

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

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

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

**wait-die** scheme (non-preemptive): Older transactions may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead. A transaction may die several times before acquiring needed data item.

**wound-wait** scheme (preemptive ): Older transaction *wounds* (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones. It may have fewer rollbacks than *wait-die* scheme.

Both in *wait-die* and in *wound-wait* schemes, a rolled back transactions is restarted with its original timestamp. Older transactions thus have precedence over newer ones, and starvation is hence avoided.

### Deadlock Detection:

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

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

If  $T_i \rightarrow T_j$  is in  $E$ , then there is a directed edge from  $T_i$  to  $T_j$ , implying that  $T_i$  is waiting for  $T_j$  to release a data item. When  $T_i$  requests a data item currently being held by  $T_j$ , then the edge  $T_i T_j$  is inserted in the wait-for graph. This edge is removed only when  $T_j$  is no longer holding a data item needed by  $T_i$ . The system is in a deadlock state if and

only if the wait-for graph has a cycle. Must invoke a deadlock-detection algorithm periodically to look for cycles.

### Deadlock Recovery

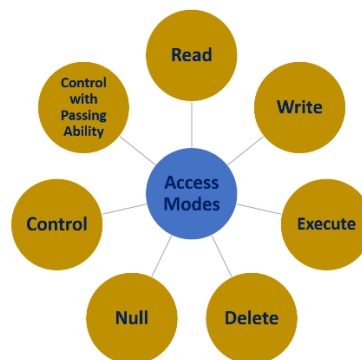
When deadlock is detected : Some transaction will have to rolled back (made a victim) to break deadlock. Select that transaction as victim that will incur minimum cost. Rollback determines how far to roll back transaction. Total rollback, aborts the transaction and then restart it. More effective to roll back transaction only as far as necessary to break deadlock. Starvation happens if same transaction is always chosen as victim. Include the number of rollbacks in the cost factor to avoid starvation.

## CHAPTER NO- 8 : DATABASE SECURITY

**27 Explain discretionary access control and mandatory access control system.**

**Ans. Discretionary Access Control**

- Allow each **user or subject** to control access to their own data.
- Following are access control.



- Each resource object on DAC has **Account Control List (ACL)**.

User/Objects	Kimsfile	Donsfile	Payrol1	Payrol2	Doesfile
Kim	RW	R	RW	R	
Joe		R			
Don		RW	R		
Jones			R		
Doe					RW
Mgr	CP	CP	C	C	C
Jim			RW	RW	

- In DAC, owner of resource restricts access to the resources based on the identity of users.

#### Advantages:

- It is Flexible.
- Simple and efficient access right management.
- Scalable means we can add more users without any complexity.

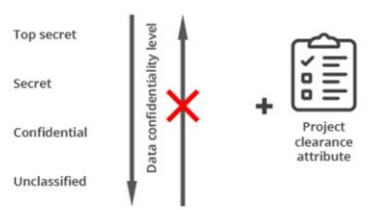
#### Disadvantages:

- Increase risk that data will be made to user that should not necessarily be given access.

- No control over information flow as one user can transfer ownership to another user.

### Mandatory Access Control

- It takes hierarchical approach to controlling access to resources.
- MAC begins with **security labels** assigned to all resources object on the system.
- Security labels are as following:



- It is generally used in Government and military services.
- Attributes are associated with classification.

EmpName	Salary	Performance	TC
Ram U	5000 C	Fair S	S
Shyam C	10000 S	Good C	S

### Advantages:

- It provides tight security because only administrator may access or alter controls.
- MAC policies reduce security errors.

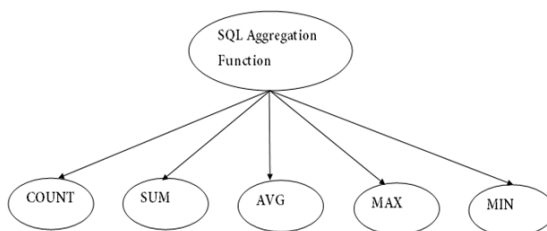
### Disadvantages:

- It require careful planning and continuous monitoring to keep all resource objects' and users' classification up to date.

## CHAPTER NO- 9 : SQL CONCEPTS

### 28 Explain Aggregate Functions in SQL.

- Ans.**
- SQL aggregation function is used to perform the calculations on multiple rows of asingle column of a table. It returns a single value.
  - It is also used to summarize the data.



- Consider this table for example:

ID	NAME	CITY	SALARY
1	TEJAS	SURAT	10000
2	AKASH	AHMEDABAD	30000
4	MANAN	RAJKOT	25000
3	DHRUVI	SURAT	20000
5	MANSI	AHMEDABAD	20000

## 1. COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(\*) that returns the count of all the rows in a specified table. COUNT(\*) considers duplicate and Null.

Syntax: COUNT(\*) or COUNT( [ALL|DISTINCT] expression )

Example 1: count no of employee.

SELECT COUNT(\*) FROM employee;

COUNT(*)
5

Example 2: count no of city.

SELECT COUNT(distinct city) FROM employee;

COUNT(distinct city)
3

## 2. SUM FUNCTION:

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax: SUM() or SUM( [ALL|DISTINCT] expression )

Example 1: select sum of salary of all employees.

Select sum(salary) from employee;

sum(salary)
105000

Example 2: select sum of salary of all employees who live in ahmedabad.

Select sum(salary) from employee where city='ahmedabad';

sum(salary)
50000

Example 2: select sum of salary of all employees city wise.(use of group by)

Select city, sum(salary) from employee group by city;

city	sum(salary)
ahmedabad	50000
surat	30000
rajkot	25000

### 3. AVG FUNCTION:

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax: AVG() or AVG( [ALL|DISTINCT] expression )

Example 1: select avg salary of employee.

```
SELECT AVG(salary) FROM employee;
```

AVG(salary)
10000

### 4. MIN FUNCTION:

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax: MIN() or MIN( [ALL|DISTINCT] expression )

Example 1: select min salary of employee.

```
SELECT MIN(salary) FROM employee;
```

MIN(salary)
10000

### 5. MAX FUNCTION:

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax: MAX() or MAX( [ALL|DISTINCT] expression )

Example 1: select max salary of employee.

```
SELECT MAX(salary) FROM employee;
```

MAX(salary)
30000

## 29 What is a view in SQL? Explain its type with syntax.

Ans.

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database. A view can either have specific rows based on certain condition or all the rows of a table. According to that there are two types of view:
  - Simple view (view is created from one table)
  - Complex view (view is created from more than one table)
- Some views are only used only for looking at table data. Other views can be used

to Insert, Update and Delete table data as well as view data. According to that there are two types of view.

1. Read – Only view
2. Updatable view.

- The reasons why views are required:
  1. When data security is required
  2. When data redundancy is to be kept to minimum while maintaining data security.

- **Creating View:**

- Syntax:

```
CREATE VIEW view_name AS
SELECT column1,
column2.....
FROM table_name WHERE [condition] [with read only];
```

Default view is created

- 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 to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM CUSTOMERS;
```

Now, you can query CUSTOMERS\_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SELECT * FROM CUSTOMERS_VIEW;
```

NAME	AGE
Ramesh	32
Khilan	25
kaushik	23
Chaitali	25
Hardik	27
Komal	22
Muffy	24

#### Deleting View

A view can be deleted using the Drop View statement.

➤ Syntax:

DROP VIEW view\_name;

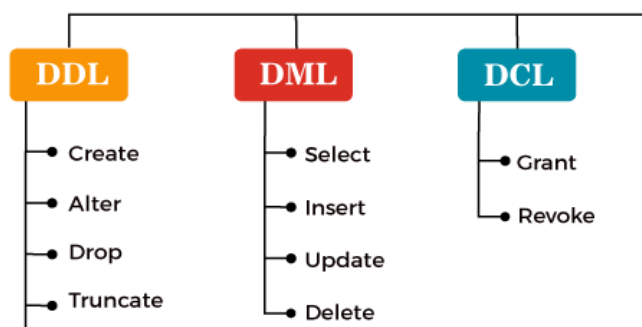
➤ Example:

DROP VIEW CUSTOMERS\_VIEW;

**30** Explain DDL, DML and DCL commands.

**Ans.**

#### Types of SQL Commands



**DDL:** DDL (Data Definition Language) statements are used to define the database structure or schema. Some examples

**1.CREATE** - to create objects in the database

CREATE TABLE <tablename>

(<ColumnName1><DataType>(<size>),<ColumnName2><DataType>(<size>), ... );

**Create table** client\_master (c\_no varchar2(5), name varchar2(10), address varchar2(20),pincode number(6), bal\_due number(10,2));

**2.ALTER** - alters the structure of the database

ALTER TABLE <TableName>



	<p>ADD (&lt;NewColumnName&gt;&lt;DataType&gt;(&lt;Size&gt;), &lt;NewColumnName&gt;&lt;DataType&gt;(&lt;Size&gt;), ...);</p> <p>Enter a new field called city in the table BRANCH_MSTR.</p> <p>ALTER TABLE BRANCH_MSTR ADD(CITY VARCHAR2(25));</p> <p>Drop the column city from the table BRANCH_MSTR.</p> <p>ALTER TABLE BRANCH_MSTR DROP COLUMN CITY;</p> <p>Alter table BRANCH_MSTR to allow the NAME field to hold maximum of 30 characters.</p> <p>ALTER TABLE BRANCH_MSTR MODIFY(NAME VARCHAR2(30));</p> <p><b>3.DROP</b> - delete objects from the database</p> <p>DROP TABLE &lt;TableName&gt;;</p> <p>Remove the table BRANCH_MSTR along with the data held.</p> <p>DROP TABLE BRANCH_MSTR;</p> <p><b>4.TRUNCATE</b> - remove all records from a table, including all spaces allocated for the records are removed</p> <p>TRUNCATE TABLE &lt;TableName&gt;;</p> <p>Truncate the table BRANCH_MSTR.</p> <p>TRUNCATE TABLE BRANCH_MSTR;</p> <ul style="list-style-type: none"> <li>• <b>DML: DML (Data Manipulation Language)</b> statements are used for managing data within schema objects</li> </ul> <p><b>1.SELECT</b> - retrieve data from the a database</p> <p>SELECT * FROM &lt;TableName&gt;;</p> <p>SELECT * FROM client_master;</p> <p><b>2.INSERT</b> - insert data into a table</p> <p>INSERT INTO &lt;tablename&gt; [(&lt;ColumnName1&gt;, &lt;ColumnName2&gt;, ... )] VALUES(&lt;value1&gt;,&lt; value2&gt;, .....);</p>
--	--

INSERT INTO client\_master (c\_no, name, address, pincode, bal\_due) VALUES ('C001', 'Alap', 'Ahmedabad', 380063, 500 );

**3.UPDATE** - updates existing data within a table

UPDATE <TableName> SET <ColumnName 1> = <Expression 1 or Value 1>, <ColumnName N> = <Expression N or Value N>;

Update the address details by changing its city name to Ahmedabad.

UPDATE ADDR\_DTLS SET City = 'Ahmedabad';

**4.DELETE** - deletes all records from a table, the space for the records remain

DELETE FROM <TableName>;

- **DCL: DCL is Data Control Language** statements. Some examples

**1.GRANT** - gives user's access privileges to database

CREATE USER Username IDENTIFIED BY PASSWORD;

GRANT CREATE SESSION TO USERNAME;

GRANT<Object Privileges>

ON<Object Name>

TO<User Name>

[WITH GRANT OPTION];

- Example:

GRANT create session to c##b41\_208;

GRANT all privileges to c##b41\_208;

**2.REVOKE** - withdraw access privileges given with the GRANT command

REVOKE<Object Privileges>

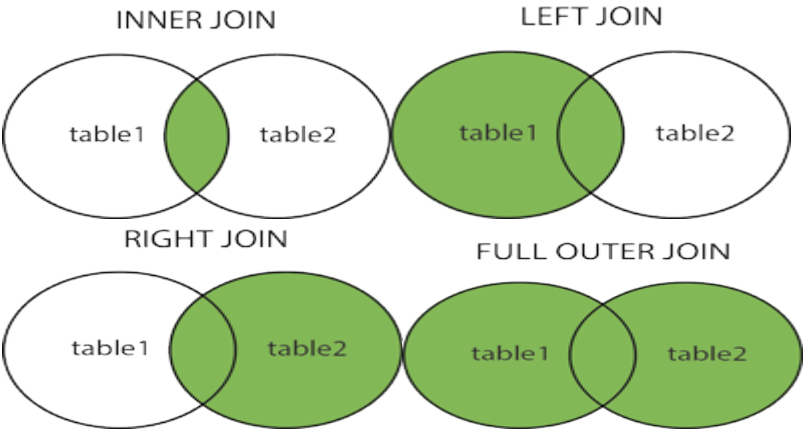
ON<Object Name>

FROM<User Name>;

Example:

REVOKE create session from c##b41\_208;

REVOKE all privileges from c##b41\_208;

31	<b>Define Join. Explain all Joins with examples in SQL.</b>
Ans.	<p>Join: A SQL Join operation is used to combine data or rows from two or more tables based on specified condition. Different types of Joins are:</p> <ol style="list-style-type: none"> <li>1. INNER JOIN:             <ol style="list-style-type: none"> <li>(a) NATURAL JOIN</li> </ol> </li> <li>2. OUTER JOIN:             <ol style="list-style-type: none"> <li>(a) LEFT OUTER JOIN</li> <li>(b) RIGHT OUTER JOIN</li> <li>(c) FULL OUTER JOIN</li> </ol> </li> </ol> <p>NATURAL JOIN: Returns records that have matching values in both tables</p> <p>LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table</p> <p>RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table</p> <p>FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table</p> <div style="text-align: center;">  </div> <p><b>INNER JOIN</b></p> <pre> SELECT table1.column1,table1.column2,table2.column1,... FROM table1 INNER JOIN table2 ON table1.matching_column = table2.matching_column; </pre> <p>Example:</p> <pre> Student(StudentID, CouseID) Courses(CouseID, CourseName) </pre>

```
SELECT Student.StudentID, Courses.CourseName
FROM Student
INNER JOIN Courses ON Student.CourseID=Courses.CourseID;
```

### **LEFT JOIN**

```
SELECT table1.column1,table1.column2,table2.column1,....
```

```
FROM table1
```

```
LEFT JOIN table2
```

```
ON table1.matching_column = table2.matching_column;
```

Example:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

### **RIGHT JOIN**

```
SELECT table1.column1,table1.column2,table2.column1,....
```

```
FROM table1
```

```
RIGHT JOIN table2
```

```
ON table1.matching_column = table2.matching_column;
```

Example:

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

### **FULL JOIN**

```
SELECT table1.column1,table1.column2,table2.column1,....
```

```
FROM table1
```

```
FULL JOIN table2
```

```
ON table1.matching_column = table2.matching_column;
```

Example:

```
SELECT Customers.CustomerName, Orders.OrderID
```

	FROM Customers FULL JOIN Orders ON Customers.CustomerID=Orders.CustomerID ORDER BY Customers.CustomerName;
32	<b>Write SQL queries for the following tables:</b> <b>T1 ( Empno, Ename , Salary, Designation)</b> <b>T2 (Empno, Deptno.)</b> (1) Display all the details of the employee whose salary is lesser than 10K. (2) Display the Deptno in which Employee Seeta is working. (3) Add a new column Deptname in table T2. (4) Change the designation of Geeta from 'Manager' to 'Senior Manager'. (5) Find the total salary of all the employees. (6) Display Empno, Ename, and Deptno. (7) Delete employee details whose Designation is 'Manager'. (8) Add new employee in T1. (9) Drop the table T1 .
Ans.	(1) Select * From T1 where Salary<10000; (2) Select Deptno From T1 inner join T2 on T1.Empno=T2.Empno and Ename='Seeta'; (3) Alter table T2 add (Deptname varchar2(25)); (4) Update T1 SET Designation='Senior Manager' Where Ename='Geeta'; (5) Select SUM(Salary) from T1; (6) Select T1.Ename, T1.Empno, T2.Deptno From T1 inner join T2 on T1.Empno=T2.Empno; (7) Delete from T1 where Designation='Manager'; (8) Insert into T1 values(3, 'seeta', 10000, 'Manager'); (9) Drop table T1;
<b>CHAPTER NO- 10 : PL/SQL CONCEPTS</b>	
33	1. Write a PL/SQL block to print the sum of even numbers from 1 to 100. 2. Write PL/SQL block to print whether the given number is Armstrong.
Ans.	1. Write a PL/SQL block to print the sum of even numbers from 1 to 100. <pre> declare     a number;     sum1 number :=0; begin     a:=1;     loop         if mod(a,2)=0 then             sum1:=sum1+a;         end if;         exit when (a=100);         a:=a+1;     end loop;     dbms_output.put_line('Sum of even numbers between 1 to 10 is '  sum1); end;</pre> 2. Write PL/SQL block to print whether the given number is Armstrong. <pre> declare     n number;     s number:=0;     r number;</pre>

```

len number;
m number;
begin
n:=&n;
m:=n;

len:=length(to_char(n));

while n>0
loop
r:=mod(n,10);
s:=s+power(r,len);
n:=trunc(n/10);
end loop;

if m=s
then
dbms_output.put_line('armstrong number');
else
dbms_output.put_line('not armstrong number');
end if;

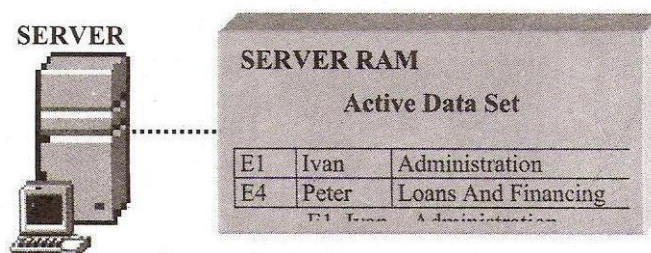
end;

```

**34 Explain cursor in PL/SQL.**

**Ans.**

- The oracle engine uses a work area for its internal processing in order to execute an SQL statements. This work area is called **CURSOR**.
- The size of the cursor in memory is the size required to hold the number of rows in the active data set.
- When user fires a select statement like:  
select emp\_no,name,dept from emp where branch\_no='b1';
- The resultant data set in the cursor opened at the server end is displayed as follows:

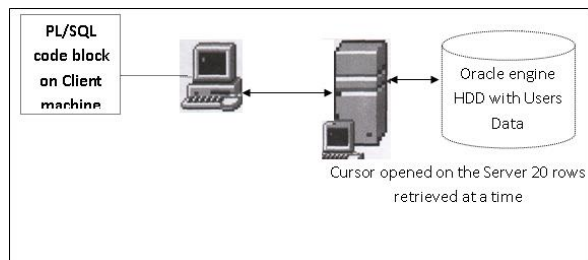


- Cursors are classified depending on the circumstances under which they are opened.
- Two types are there:

- Implicit cursor
- Explicit cursor
- Both implicit and explicit cursor have four attributes as below:

ATTRIBUTE NAME	DESCRIPTION
%ISOPEN	Returns TRUE if cursor is open, FALSE otherwise
%FOUND	Returns TRUE if record was fetched successfully, FALSE otherwise
%NOTFOUND	Returns TRUE if record not fetched successfully, FALSE otherwise
%ROWCOUNT	Returns number of records processed from the cursor

- **Implicit Cursor**
- The oracle engine implicitly opens a cursor on the server to process each SQL statement.



- It is default cursor in PL/SQL.
- Implicit cursor attributes can be used to access information about the last insert, update and delete or single row select statements.

ATTRIBUTE NAME	DESCRIPTION
%ISOPEN	The oracle engine automatically opens and close the SQL cursor after executing its associated select, insert, update or delete SQL in implicit cursor. Thus the SQL%ISOPEN attribute can't be referenced outside the SQL statement. So, it is always evaluates FALSE.
%FOUND	Returns TRUE if record was fetched successfully, FALSE otherwise. SYNTAX IS SQL%FOUND.
%NOTFOUND	Returns TRUE if record not fetched successfully, FALSE otherwise. SYNTAX IS SQL%NOTFOUND.

**%ROWCOUN  
T**

Returns number of rows affected by an insert, update or delete or select statement. SYNTAX IS SQL%ROWCOUNT.

### **Explicit cursor**

- Explicit cursor are used when you are executing a SELECT statement that will return more than one row.
- Explicit cursor attributes can be used to access information about the last insert, update and delete or single row select statements.

Attribute Name	Description
%FOUND	Evaluates to true, if the last fetch succeeded because a row was available; or to false, if last fetch failed. The syntax is CursorName%found.
%NOTFOUND	It is logical opposite of %found. The syntax is CursorName%found.
%ISOPEN	It is TRUE if cursor is OPEN, otherwise FALSE. The syntax is CursorName%open.
%ROWCOUNT	Returns the no of row fetched from the active data set. It is set to zero when cursor is opened. The syntax is CursorName%rowcount.

### **CURSOR DECLARATION:**

#### **Syntax:**

CURSOR CursorName IS SELECT statement;

Declaration is done in declare section.

Memory is not allocated at this point.

Three commands used to control the cursor subsequently are

1. open
2. fetch
3. close

#### **1. OPEN THE CURSOR:**

##### **Syntax:**

OPEN CursorName;

- Initialization of a cursor takes place via the OPEN statement:
- Memory is allocated.

#### **2. FETCHING RECORD FROM THE CURSOR:**

##### **Syntax:**

FETCH CursorName INTO Variable1, Variable2, .....;

- Retrieve data into local variable.
- Access record from cursor but one row at a time.

#### **3. CLOSING CURSOR:**

**Syntax:** CLOSE CursorName



35	<b>Write a note on stored procedure with suitable example.</b>
Ans.	<p>The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. These subprograms do not return a value directly; mainly used to perform an action. It is just like procedures in other programming languages.</p> <p>The procedure contains a header and a body.</p> <ul style="list-style-type: none"> <li>➤ Header: The header contains the name of the procedure and the parameters or variables passed to the procedure.</li> <li>➤ Body: The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.</li> </ul> <p><b>Syntax:</b></p> <pre>CREATE [OR REPLACE] PROCEDURE procedure_name [(parameter_name [IN   OUT   IN OUT] type [, ...])] {IS   AS} BEGIN     &lt; procedure_body &gt; END;</pre> <ul style="list-style-type: none"> <li>➤ procedure-name specifies the name of the procedure.</li> <li>➤ [OR REPLACE] option allows the modification of an existing procedure.</li> <li>➤ The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.</li> <li>➤ Procedure-body contains the executable part.</li> <li>➤ The AS keyword is used instead of the IS keyword for creating a standalone procedure.</li> </ul> <p><b>How to execute:</b></p> <p>A procedure can be called in two ways –</p> <ul style="list-style-type: none"> <li>• Using the <b>EXECUTE</b> keyword</li> <li>• Calling the name of the procedure from a PL/SQL block</li> </ul> <p><b>Example:</b></p> <pre>CREATE OR REPLACE PROCEDURE greetings AS BEGIN     dbms_output.put_line('Hello World!'); END;</pre> <p>/</p> <p>The above procedure named '<b>greetings</b>' can be called with the EXECUTE keyword as –</p> <pre>EXECUTE greetings;</pre> <p>The above call will display –</p>

	<p>Hello World</p> <p>PL/SQL procedure successfully completed.</p>
<b>36</b>	<b>What is trigger and need of trigger?</b>
<b>Ans.</b>	<p>Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events</p> <ul style="list-style-type: none"> <li>• A <b>database manipulation (DML)</b> statement (DELETE, INSERT, or UPDATE)</li> <li>• A <b>database definition (DDL)</b> statement (CREATE, ALTER, or DROP).</li> <li>• A <b>database operation</b> (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).</li> </ul> <p><b>Syntax:</b></p> <pre> <b>CREATE</b> [OR REPLACE ] <b>TRIGGER</b> trigger_name { BEFORE   <b>AFTER</b>   <b>INSTEAD OF</b> } { <b>INSERT</b> [OR]   <b>UPDATE</b> [OR]   <b>DELETE</b> } [<b>OF</b> col_name] <b>ON</b> table_name [<b>REFERENCING</b> OLD <b>AS</b> o NEW <b>AS</b> n] [<b>FOR EACH ROW</b>] <b>WHEN</b> (condition) <b>DECLARE</b>     Declaration-statements <b>BEGIN</b>     Executable-statements <b>EXCEPTION</b>     Exception-handling-statements <b>END;</b> </pre> <ul style="list-style-type: none"> <li>➤ <b>CREATE [OR REPLACE] TRIGGER trigger_name:</b> It creates or replaces an existing trigger with the trigger_name.</li> <li>➤ <b>{ BEFORE   AFTER   INSTEAD OF }:</b> This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.</li> <li>➤ <b>{ INSERT [OR]   UPDATE [OR]   DELETE }:</b> This specifies the DML operation.</li> <li>➤ <b>[OF col_name]:</b> This specifies the column name that would be updated.</li> <li>➤ <b>[ON table_name]:</b> This specifies the name of the table associated with the trigger.</li> <li>➤ <b>[REFERENCING OLD AS o NEW AS n]:</b> This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.</li> </ul>

- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

**Advantages and need of trigger:**

- Triggers can be used for implementing referential integrity constraints.
- By using trigger, business rules and transactions are easy to store in database and can be used consistently even if there are future update to the database.
- It control updates allowed in database.
- When a change happens in a database a trigger can adjust the change to the entire database.
- Triggers are used for calling stored procedure.

**Types of trigger:**

There are two types of triggers based on the level on which it is triggered.

1. **Row level trigger:** An event is triggered for each row updated, inserted or deleted.
2. **Statement level trigger:** An event is triggered for each SQL statement executed.