Unit-2 Relational Data Model

Basic Concepts of Relational Model

- Simplest and widely used data Model
- Relational Model was proposed by E.F. Codd in 1970
- Data is represented in the form of relations or tables
- Table/relation is a set of rows and columns

Roll_No.	Name	Department
101	Steive	Comp. Sci.
265	Jhoson	Finance
505	Margret	Biology
325	Jenny	Social Sci.
256	Davis	Comp. Sci.
453	Sheryl	Biology
365	Emma	Maths

Student Relation in Relational Model

Key concepts of Relational model

- Attribute: Each relation is defined in terms of some properties each of which is known as attribute
- Domain : The possible values an attribute can take in a relation is called its domain.
- Tuple: Each row of a relation is known as tuple
- NULL values: Values of some attribute for some tuple may be unknown, missing or undefined which are represented by NULL
- Degree : The number of attributes of the relation is known as degree of relation

Codd's Rules of DBMS

- Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.
- Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

- Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

- Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following data is missing, data is not known, or data is not applicable.

Codd's Rules of DBMS

- Rule 4: Active Online Catalog

The structure and description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

- Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

- Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Codd's Rules of DBMS

- Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

- Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

- Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

Codd's Rules of DBMS

- Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

- Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

- Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

Tables used

- Following tables will be used in examples .

1. dept (deptno,dname,location)
2. emp(empno,ename,job,salary,deptno)

3. customers(customer_id,customer_name,address,city,postalcode, country)

Structured Query Language(SQL)

- An open source Domain Specific language developed to create, maintain and retrieve data from relational databases, e.g. MySQL, Oracle, SQL server etc.
- Used to work with Structured data
- Not a case sensitive language. But in routine practice keywords are written using Capital letters and user defines values are written in small letters
- The data types and syntax in SQL may slightly vary from vendor to vendor
- Highly scalable and flexible
- Can manage numerous transactions and heavy workload
- Provides high security through various authorization constraints
- It works for every small or large organization

SQL Data Types

1. Numeric data type

Integer types: INTEGER, SMALL INT, TINY INT, MEDIUM INT, BIG INT

Fixed point values: DECIMAL

Floating point values: FLOAT, DOUBLE

2. Character data type

CHAR, VARCHAR

BINARY, VARBINARY

BLOB, TEXT

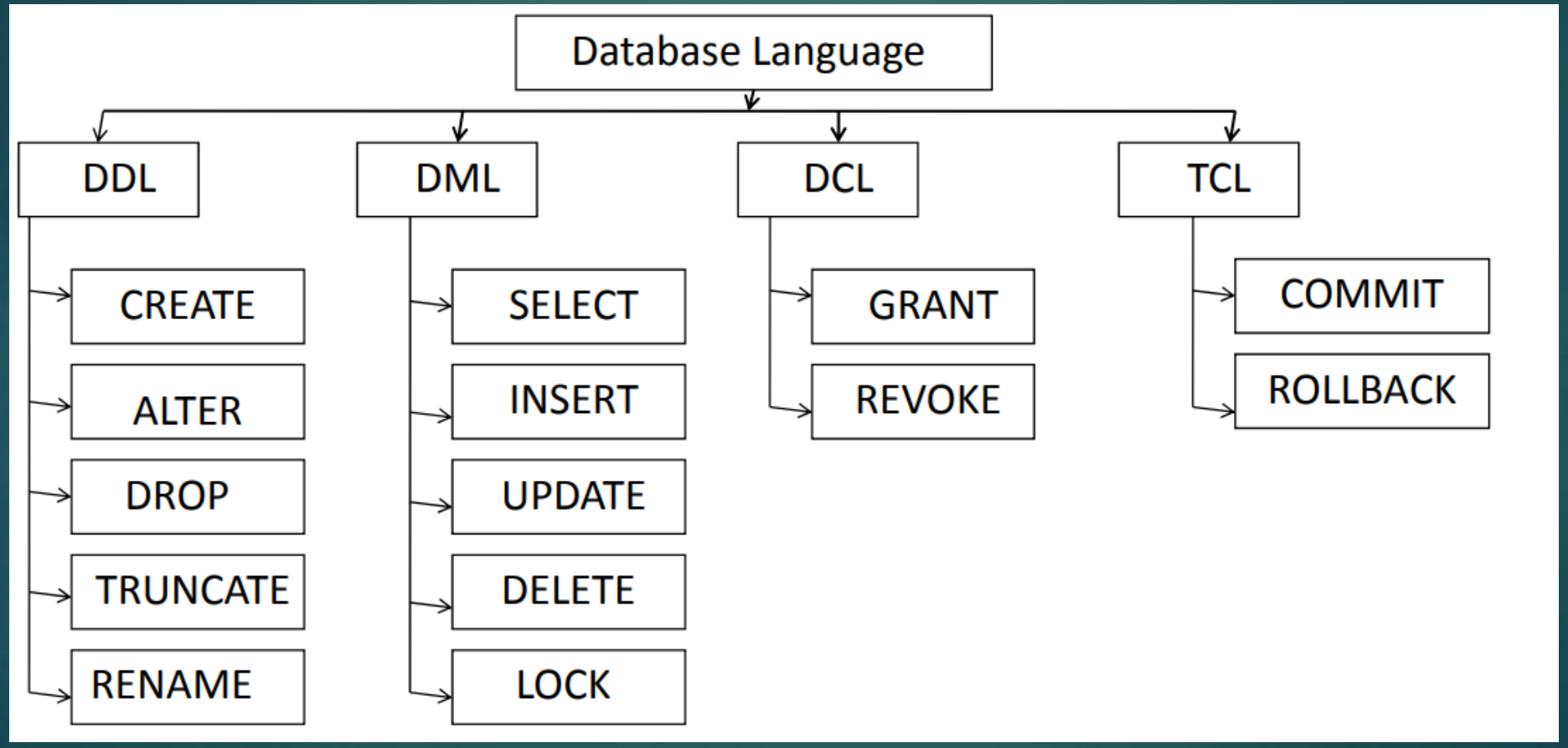
3. Date and Time data type

DATE , TIME

DATETIME, TIMESTAMP

SQL Languages

Database language provides an interface to connect to the database



Data Definition Language(DDL)

- DDL is used for defining the database schema(structure of database objects)
- It is used for creating tables, views, indexes, constraints etc. in database. Commands under DDL are

1. Create: It is used to create objects in the database
2. Alter: It is used to alter the structure of the database e.g. Add, delete or modify column
3. Drop: It is used to delete objects from the database
4. Truncate: It is used to remove all records from a table
5. Rename: It is used to rename an object

CREATE TABLE statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (
  column1 datatype,
  column2 datatype,
  column3 datatype,
 ....
 );
```

```
Eg. Create table dept(
   deptno integer primary key,
   dname varchar(15) not null,
   location varchar(15));

   create table emp(
   empno integer primary key,
   ename varchar(15) not null,
   job varchar(15),
   salary number(10),
   deptno integer references dept(deptno));
```

SQL ALTER TABLE Statement

The `ALTER TABLE` statement is used to add, delete, or modify columns in an existing table.
The `ALTER TABLE` statement is also used to add and drop various constraints on an existing table.

To add a column in a table

- ALTER TABLE *table_name*
  ADD *column_name datatype*;
- Eg. ALTER TABLE Customers
  ADD Email varchar(255);

To delete a column in a table

- ALTER TABLE *table_name*
  DROP COLUMN *column_name*;
- Eg. ALTER TABLE Customers
  DROP COLUMN Email;

To change the data type of a column in a table, use the following syntax:

- ALTER TABLE *table_name*
  MODIFY COLUMN *column_name datatype*;
- Eg. ALTER TABLE Persons
  MODIFY COLUMN email varchar(150);

The DROP TABLE statement is used to drop an existing table in a database.
Syntax
DROP TABLE *table_name*;
Eg. Drop table dept;

The TRUNCATE TABLE command deletes the data inside a table, but not the table itself.
Eg.
TRUNCATE TABLE Categories;

RENAME command is used to set a new name for any existing table.

syntax.
RENAME TABLE old_table_name to new_table_name

Eg. RENAME TABLE emp to employee;

TRUNCATE TABLE

TRUNCATE TABLE command deletes the data inside a table, but not the table itself.

The following SQL truncates the table "Categories":

Eg.

```
TRUNCATE TABLE Categories;
```

RENAME TABLE

RENAME table command is used to rename or change the name of existing table.

Eg. Rename emp to employee

Data Manipulation Language(DML)

- DML is used for accessing and manipulating data in a database. Commands which comes under DML are

1. Select: It is used to retrieve data from a database
2. Insert: It is used to insert data into a table
3. Update: It is used to update existing data within a table
4. Delete: It is used to delete all records from a table

DML Commands

SELECT command is used to select data from a database table
Syntax: select */col 1,col 2 ,col 3 from table-name;
Where * means all columns , col1 means column1 , and so on

Eg. select * from emp;
select empno,ename,salary from emp;

The `SELECT DISTINCT` statement is used to return only distinct (different) values.
Select distinct country from customers;
(to select distinct country and to eliminate duplicate country)

Insert command : `INSERT INTO` statement is used to insert new records in a table
Syntax: insert into table-name values (value1,value2,value3,……);
Eg. insert into dept values(10,'Finance','Andheri');

Syntax: insert into table-name(col1,col2,clo3,……)
values(value1,value2,value3);
Eg. insert into dept(deptno,dname) values(20,'Purchase');

Update command : it is used to modify the existing records in a table.

syntax: UPDATE *table_name* SET *column1 = value1, column2 = value2, ...*WHERE *condition*;

eg. update emp set salary=40000, job='senior clerk'
where job='clerk';

Delete Command : it is used to delete existing records from table.

syntax : delete from table_name where condition;

eg. Delete from emp where job='clerk';

DCL and TCL

- DCL is used to Grant and Revoke access on database

1. Grant: It is used to give user access privileges to a database

   eg. grant select,update on emp to pooja;

   grant all on emp to pooja;

2. Revoke: It is used to take back permissions from the user

   eg. revoke all on emp from pooja;

   revoke select ,update on emp from pooja;

- Transaction Control Language(TCL)

TCL is used to execute changes done by DML commands

1. Commit: It is used to save the transaction on the database

2. Rollback: It is used to restore the database to original since the last Commit

Integrity Constraints

- Integrity refers to accuracy, completeness and consistency of data in the database
- Constraints are Set of rules/restrictions that are enforced on attributes of relation to maintain consistency
- Constraints are used to ensure correctness and accuracy of data stored in database
- Allows user to determine what data should be part of database, what kind of modifications are allowed

Types of Integrity Constraints

- Domain Integrity Constraints
- Entity Integrity Constraints
- Referential Integrity Constraints

Domain Integrity Constraints

- It restricts the type of values a column or relation can hold in the database

- NOT NULL: It ensures that a column will never hold a NULL value

  e.g. Create table student (roll_no integer NOT NULL)

- DEFAULT : Specifies a default value for a column if no valued is given while inserting tuple

  e.g. Create table student (dept varchar(10) DEFAULT ‘Computer’)

- CHECK : It is used to specify the range of value that can be inserted in any column

  e.g. create table voter_id(age integer CHECK (age>=18))

Entity Integrity Constraints

- Entity Integrity Constraint is used to ensure the uniqueness of each record or row in the data table
- PRIMARY KEY : Field in the table which uniquely identify each record in the table.Does not allow duplicate values and NULL value

  e.g. Create table employee (empno integer Primary key)

- UNIQUE : It ensures that a column will have unique value for all tuples in a relation. It allows NULL value

  e.g. Create table employee (phoneno integer Unique)

Referential Integrity Constraints

- Referential Integrity Constraint ensures that there always exists a valid relationship between two tables. This makes sure that if a foreign key exists in a table relationship then it should always reference a corresponding value in the second table or it should be null
- Foreign Key : It is an attribute that used to link two different tables.It is a common attribute between two tables

e.g. create table employee (empno integer primary key,

ename varchar(10),deptno integer references dept(deptno))

Table level and Column level Constraints

Table level Constraint	Column level Constraint
table level constraints apply to the whole table	Column level constraints apply to a column
Applicable to multiple columns. i.e. Used to define composite keys	Applicable to single column
NOT NULL constraint is not allowed at table level	NOT NULL constraint is allowed at column level
e.g. Create table order(pid integer PRIMARY KEY,pname NOT NULL, cid integer, order_date date, FOREIGN KEY (cid) references cust(cid));	e.g. Create table borrower(loan_no integer PRIMARY KEY, amount integer NOT NULL);

Clauses in SQL

Where clause : WHERE clause is used to filter records.It is used to extract only those records that fulfill a specified condition.

- Syntax

```
SELECT column1, column2, ...
    FROM table_name
     WHERE condition;
```

e.g. SELECT * FROM Customers WHERE Country='Mexico';

Group by Clause

- The **Group By** clause is used for organizing similar data into groups. It means, if different rows in a precise column have the same values, it will arrange those rows in a group.

Syntax :
SELECT column1, function_name(column2) **FROM** table_name

WHERE condition **GROUP BY** column1;

e.g. Select deptno,sum(sal) from emp

group by deptno;

Select deptno,count(empno) from emp

group by empno;

Having Clause

The HAVING clause was added to SQL because the WHERE clause cannot be used with aggregate functions .It is used to filter records after group by clause

Syntax. SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
HAVING *condition*

- *e.g.* SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  HAVING COUNT(CustomerID) > 5;

Order by Clause

The ORDER BY clause is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

- Syntax. SELECT column1, column2, ...
  FROM table_name
  ORDER BY column1, column2, ... ASC | DESC;

- e.g. SELECT * FROM Customers
  ORDER BY Country;

Nested Queries in SQL

- Nested query is a query written/embedded within another query. It is also called as subquery
- It is written within the parenthesis after from clause in outer/main/enclosing query
- The result of inner/nested query is given as a input to outer query for execution
- Group by and Order by clause can not be used in subquery
- Between operator can not be used with subquery.

Nested Query Examples

List all employees whose salary is same as that of 'John'

```
select * from emp where salary=(select salary from emp
where ename='John');
```

List the details of employees who are working in same department as that of 'Ram'

```
select * from emp where deptno=(select deptno from emp
where ename='Ram');
```

Joins in SQL

JOIN is used to combine data or records from two or more than two tables. In order to access data from two tables it requires some common field between them

Different types of joins are :

1. (INNER) JOIN /Equi join):
   Returns records that have matching values in both tables
   eg: select empno,ename,dept.deptno,dname,location from emp inner join dept on emp.deptno=dept.deptno;

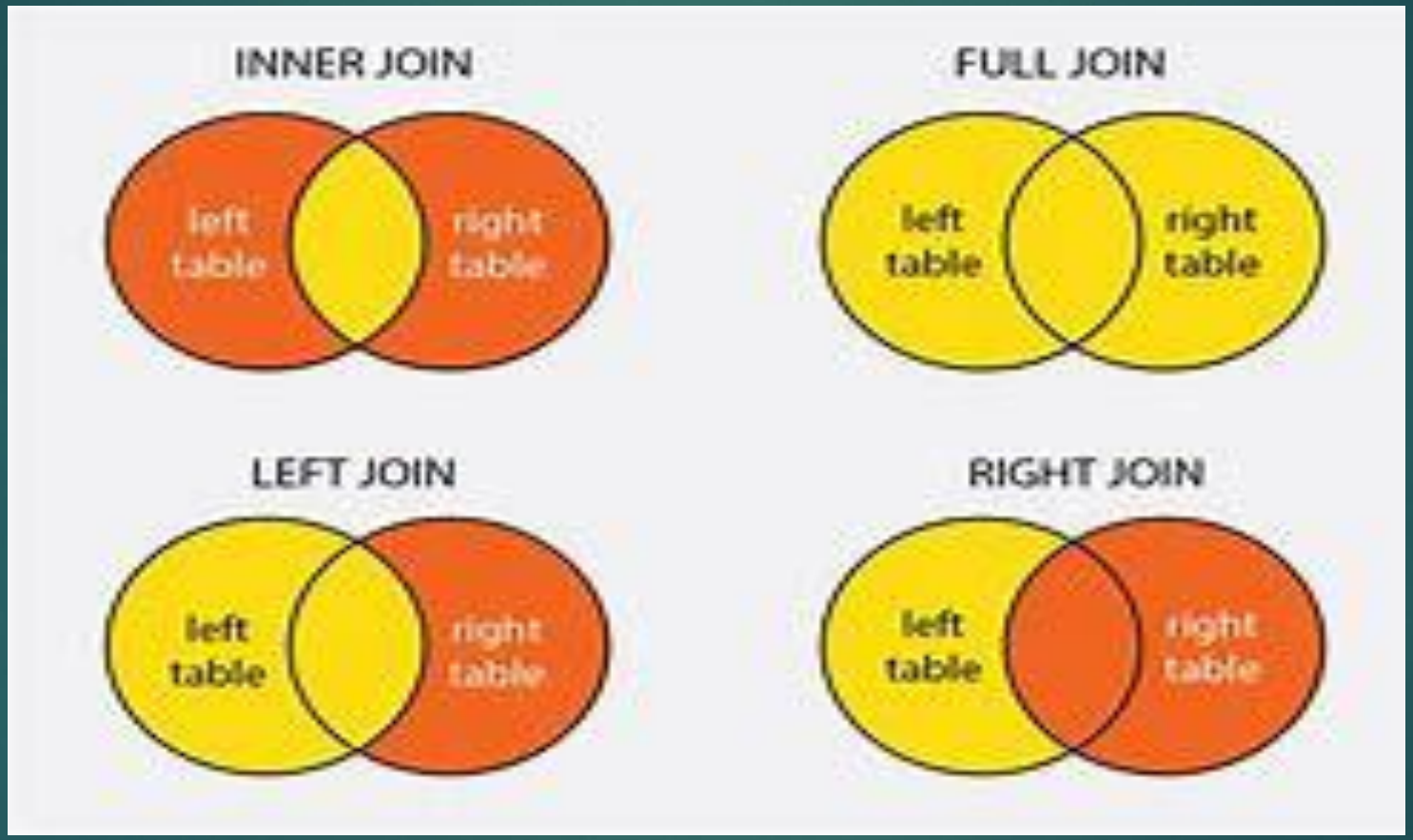
2. Non Equi Join : uses comparison operators(IN,BETWEEN,!=,>,=,<=) other than equal(=) in join condition.
   eg. select empno,ename,dept.deptno,dname,location from emp join dept on emp.deptno>dept.deptno;

3. Self Join : In SELF JOIN table is joined with itself
   eg. Select a.empno,a.ename,a.salary ,b.empno,b.ename,b.salary from emp a,emp b where a.empo != b.empno;

4. LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
   eg. select empno,ename,dept.deptno,dname,location from emp
   left outer join dept on emp.deptno=dept.deptno;

5. RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
   eg. select empno,ename,dept.deptno,dname,location from emp right
   outer join dept on emp.deptno=dept.deptno;

6. FULL (OUTER) JOIN: Returns all records matching in both tables and also non matching records from both tables
   eg. select empno,ename,dept.deptno,dname,location from emp
   full outer join dept on emp.deptno=dept.deptno;


INNER JOIN
left table
right table
FULL JOIN
left table
right table
LEFT JOIN
left table
right table
RIGHT JOIN
left table
right table

Views in SQL

- View is a Virtual Table which is created from original table/tables in database
- View can be created by fetching attributes from one or more tables
- Views are created in order to simplify complex queries and to maintain security by providing restricted access to database
- It may consist of either all rows of a original table or only limited rows that satisfy condition specified in query
- View does not have physical existence i.e. It does not occupy physical memory(except Materialized view),Only the query creating view is stored in data dictionary
- View is created dynamically each time user executes query using view name.

Types of Views

- 1. **Simple View**: It is created on a single table.It does not contain any function or Group By clause.

  eg. Create view empview as

  select empno,salary from emp where salary>50000;

- 2. **Complex View**: It is created from by fetching data from one or multiple tables. It contains functions, JOIN conditions, Order by, Group by clause.

  eg. Create view emp_dept as

  select deptno,count(empno) from emp group by deptno;

- 3. **Materialized View**: It is not a virtual table. It is a physical copy of original(base) table. It actually store result of query. i.e. It occupy physical memory. It takes less time to execute queries. used in data warehousing. Materialized view reduce the processing time to regenerate the whole data. It helps remote users to replicate data locally and improve query performance.

  eg. Create materialized view empview as select dept.deptno,dname,

  location ,sum(salary),avg(salary) from dept,emp group by dept.deptno;

Operations on Views

- All DML queries can be executed on View
- On complex view DML queries can not be applied directly. Need to write function/trigger
- The syntax of queries for view is same as that queries we execute on table
- The changes/updating done on base/original table is reflected in view also and vice versa
- At the time of view updating, only those attributes can be updated which are selected while creating view
- All constraints applied on base/original table are also reflected in View
- JOIN operations can also be performed on views

Relational Algebra

- Relational algebra is a procedural query language, which takes instances of relations as input and gives instances of relations 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
- Project
- Union
- Set difference
- Cartesian product
- Rename

- Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation.

p is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

- $\sigma_{subject = "database"}(Books)$

Output − Selects tuples from books where subject is 'database'.

- $\sigma_{subject = "database" \text{ and } price = "450"}(Books)$

Output − Selects tuples from books where subject is 'database' and 'price' is 450.

- Project Operation (∏)

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

Notation – $\prod_{A1}, A_2, A_n (r)$

Where A_1, A_2, A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated

For example – $\prod_{subject, author}$ (Books)

Selects and projects columns named as subject and author from the relation Books.

- Union Operation (∪)

It performs binary union between two given relations and is defined as –

Notation : r U s

Where **r** and **s** are database relations ,For a union operation to be valid, the following condition must hold –

r, and **s** must have the same number of attributes.

Eg. $\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

Output – Projects the names of the authors who have either written a book or an article or both

- Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation : **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

Eg. $\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Output − Provides the name of authors who have written books but not articles

- Cartesian Product (X)

Combines information of two different relations into one. Cross product between two relations let say A and B, (A X B) will results all the attributes of A followed by each attribute of B. Each record of A will pairs with every record of B.

Notation : r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

$\sigma_{author = 'tutorialspoint'}$(Books X Articles)

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation : $\rho_x(E)$

Where the result of expression **E** is saved with name of **x**.

Key Concepts in SQL

- Primary key: It is an attribute or field in a relation which uniquely identifies each record of a relation.
- Super Key: A super key is a set of one or more attributes (columns), which can uniquely identify a row in a table.
- Candidate Key: Minimal Super key is candidate key. there is one and only one primary key in any relationship but there is more than one candidate key . Candidate key's attributes can contain a NULL value which opposes to the primary key.
- Foreign Key:A foreign key is a column or attribute in a table that provides a link between data in two tables. It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them.

THANK - YOU