Module 4:
Structured Query Language (SQL)

4.1 Outline:

- Overview of SQL
- Data Definition Commands
- Integrity constraints
  - Key constraints
  - Domain Constraints
  - Referential integrity
  - Check constraints
- Data Manipulation commands
- Data Control commands

Overview of SQL

- SQL stands for **Structured Query Language**
- Used to communicate with a database
- Standard language for relational database management systems
- SQL statements are used to perform different operations on database like retrieval, insertion, updation and deletion of data
- Some common RDBMS that use SQL are MySQL, Oracle, Microsoft Access, Microsoft SQL Server, Sybase, Ingres, etc.
- SQL is developed by IBM as a part of System R project in 1970

Overview of SQL

- Initially it was called as Sequel
- SQL was one of the first commercial languages for Edgar F. Codd's relational model.
- The steps required to execute SQL statements are handled transparently by the SQL database.
- SQL can be characterized as non-procedural

As **procedural languages** the details of the operations to be specified, such as opening and closing tables, loading and searching indexes, and writing data to file systems are required which is not necessary in SQL.

Characteristics of SQL

- SQL is an ANSI and ISO standard computer language for creating and manipulating databases.
- SQL allows the user to create, update, delete, and retrieve data from a database.
- The tokens and syntax of SQL are oriented from English common speech to keep the access barrier as small as possible.

  Hence it is very simple and easy to learn.
- All the keywords of SQL can be expressed in any combination of upper and lower case characters.

  It makes no difference whether UPDATE, update, Update, UpDate i.e. the keywords are case insensitive

Characteristics of SQL

- SQL is very powerful language
- SQL works with database programs like DB2, Oracle, MS Access, Sybase, MS SQL Sever etc.

Advantages of SQL

1. High Speed
2. Portable
3. Well Defined Standards Exist
4. Supports object based programming
5. Used with all DBMS systems with any vendor
6. No Coding Required
7. Used for relational databases

Advantages of SQL

8. Easy to learn and understand
9. Complete language for a database
10. Dynamic database language
11. Can be used as programming and interactive language
12. Client/Server language
13. Multiple data views
14. Used in internet

Data types in SQL

- In SQL, we store the data in tabular format where table (relation) is the combination of rows (tuples) and columns (fields).
- While creating table we have to assign data types to the columns.
- These data types are used to decide that which type of data the columns can store.

Data types in SQL

Numeric Data Types

- This data type is used to store a number values that can be decimal or floating point values

a) **Integer number of various sizes**:

- These types of system are used to store natural numbers which are not having any decimal values.
- Example 111, 23 etc.
- As per size of number we can use following types of integers:

i. Interger(p)

ii. Integer or INT

iii. SMALL INT

iv. BIGINT

Data types in SQL

b) Floating point numbers of various precision

- This system is used for storing decimal numbers which may be of greater size than integers.
- Example 11.2, 12.3 etc.
- As per size of floating point number we can use following types of numbers.
  - i) FLOAT or REAL
  - ii) DOUBLE PRECISION

Data types in SQL

c) **Formatted numbers**

- This system used for storing some special numbers which may be of greater size than integers and floating point numbers.

i. DECIMAL or DEC (i, j)

where

- i = Precision = Total number of digits in number.
- j Scale = Total number of digits after decimal point. (default value is 0)
- 12.234 (Decimal(5,3))

ii. NUMERIC (i, j)

- Example 1.12342 (Numeric(1,5))

Data types in SQL

Character string data type

- This data type is used to store a character string which is combination of some alphabets and enclose single quotation marks.
- Example: 'Mahesh', 'abc' etc.

a) **Fixed length:** CHAR (n), Where n= number of characters.

Example abc is stored in char (10) will be stored as 'abc, (abc padded with 7 blank spaces)

b) **Varying length** : VARCHAR (n) Where n = maximum number of characters.

Example It'abc' is stored in VARCHAR (10) will be stored as 'abc ‘(no blank spaces).

Data types in SQL

- **Date time data type**

a) **Date**

- The date data type has 10 positions and its components are YEAR, MONTH and DAY in form in form YYYY-MM-DD
- The length is 10.
- Example Date ‘2009-01-01' (as ‘YYYY-MM-DD’)

b) **Time**

- The TIME data type has at least eight positions, and its components are HOUR, MINUTES and SECONDS in HH:MM:SS
- Example Time 11:16:59' (as HH:MM:SS)

Data types in SQL

c) **Timestamp/ date time**

- The TIMESTAMP data type includes both date and time fields
- Represented using the fields YEAR, MONTH, DAY, HOUR, MINUTE and SECOND in the format YYYY-MM DD HH:MM:SS
- Example
  - Timestamp 2009:01:01 11:16:59 648302 (as 'YYYY-MM-DD HH:MM:SS TIMEZONE')
  - CurrentTimeStamp: Local date and time without time zone.

Data types in SQL

d) **Interval**

- This specifies an interval a relative value that can be used to increment or decrement an absolute value of date, time or timestamp.
- INTERVAL YEAR TO MONTH Datatype.
- Example:
- **‘21-5’ Year(2) To Month** indicates an interval of 21 years and 5 months
- **‘21-5’ Year(2)** indicates an interval of 21 years

SQL Languages

Types of SQL Commands			
DDL	**DML**	**DCL**	**TCL**
CREATE	SELECT		
ALTER	INSERT	GRANT	COMMIT
DROP	UPDATE	REVOKE	ROLLBACK
TRUNCATE	DELETE		SAVEPOINT
RENAME	MERGE		

Data Definition Language(DDL)

- To create database schema and database objects like table, Data Definition Language is used
- DDL statements are used to build and modify the structure of your tables and other objects in the database
- Set of DDL Commands:
  - **CREATE Statement**: to create database objects
  - **ALTER Statement**: to modify structure of database objects
  - **DROP Statement**: to remove database objects
  - **RENAME Statement**: to rename database objects
  - **TRUNCATE Statement**: to empty database tables
- Database objects are any data structure created in database (Tables, Views)

Data Definition Language(DDL)

- When you execute a DDL statements, it takes effect immediately, as it is **auto-committed** into database

Hence no rollback(undo) can be performed with these set of commands

CREATE Command

- CREATE statement is used to create new database objects like table, index and others
- The CREATE TABLE statement is used to create new table with unique name

```
Syntax : -
CREATE TABLE table_name
(
Column_name1 data_type (size) [constraints],
Column_name2 data_type (size) [constraints],
Column_name3 data_type (size) [constraints]
);
```

DESC Command

- DESC command is used to display structure of the table
- **Syntax**:

DESCRIBE TableName;

OR

DESC TableName;

CREATE Command

Example

```
SQL> CREATE TABLE Employee
     (    Eid  INT,
     Name     VARCHAR (20),
     Age      INT,
     Address  CHAR (25),
     Salary   DECIMAL (18, 2)
     );
Query OK, 0 rows affected (0.01 sec)
```

To view the structure of newly created table.

```
SQL> DESC Employee;
+---------+----------------+------+-----+---------+-------+
| Field   | Type           | Null | Key | Default | Extra |
+---------+----------------+------+-----+---------+-------+
| EID     | int(10)        | YES  |     | NULL    |       |
| NAME    | varchar(20)    | YES  |     | NULL    |       |
| AGE     | int(11)        | YES  |     | NULL    |       |
| ADDRESS | char(25)       | YES  |     | NULL    |       |
| SALARY  | decimal(18,2)  | YES  |     | NULL    |       |
+---------+----------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

CREATE Command

CREATE Table from Existing Table

```
CREATE TABLE new_table
  AS (SELECT column_1, column2, ... column_n
      FROM old_table);
```

Example: CREATE Table from Existing Table

```
CREATE TABLE suppliers
  AS (SELECT id, address, city, state, zip
      FROM companies
      WHERE id > 1000);
```

ALTER Command

- Once database object is created in database, we may require ALTER command to update structure of database object
- The ALTER Statement can be used to add, delete or modify columns in existing table

```
ALTER TABLE table_name
ADD column_name  datatype

ALTER TABLE table_name
DROP COLUMN column_name

ALTER TABLE table_name
ALTER MODIFY column_name  datatype
```

ALTER Command

```sql
ALTER TABLE Customers
ADD phone varchar(10);
```

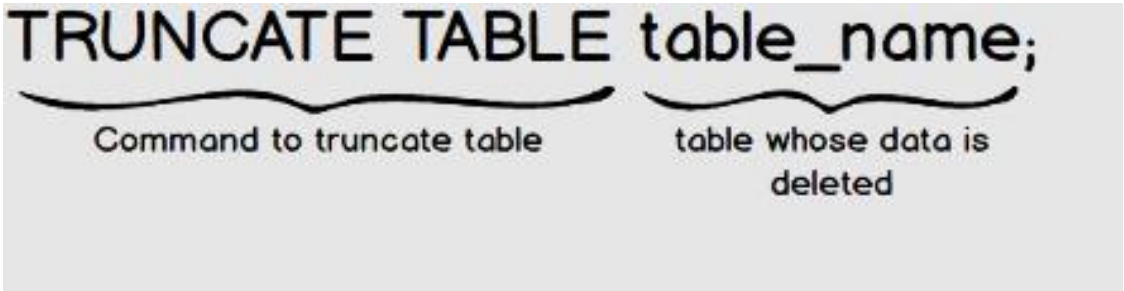
```sql
ALTER TABLE Customers
ADD phone varchar(10), age int;
```

```sql
ALTER TABLE Customers
MODIFY COLUMN age VARCHAR(2);
```

```sql
ALTER TABLE Customers
DROP COLUMN age;
```

TRUNCATE Command

- This command is used to **delete all records from existing table**
- **It is possible to do same action with DROP TABLE command but it would remove complete table structure from the database**
- A DELETE command will also remove all data from table but with DELETE data deletion can be rolled back and truncate acts as permanent data deletion with no roll back possible
- Truncate will de-allocates memory space. So that free space can be used by other tables unlike DELETE command



DROP Command

- DROP command can be used to remove database objects from user database
- DROP TABLE statement is used to remove table definition and all related data like indexes, triggers, constraints and permissions for the table
- Developer must be careful while running this command because once table is dropped then all information available in that table will also be lost forever and no rollback can be done

Syntax	DROP TABLE <TABLENAME>;
Sql Command	DROP TABLE EMPLOYEE;

Rename Command

- It is possible to change name of table with or without data in it using simple RENAME command
- We can rename table object at any point in time

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

```
ALTER TABLE table_name
CHANGE COLUMN old_name TO new_name;
```

Data Manipulation Language(DML)

- Data Manipulation Language statements are used for manipulating data in database
- DML commands are not auto-committed like DDL Statements Changes done by DML commands can be rolled back
- Under DML Commands we perform:
  - **INSERT Statement**
  - **UPDATE Statement**
  - **DELETE Statement**
  - **SELECT Statement**

Insert Statement

- INSERT Statement used to add records to existing table
- To insert data into table, SQL INSERT INTO command can be used
- To insert few values in table as per columns names we can use following
- **Syntax:**

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Insert Statement

- If all values for all the columns of the table are to be added then **no need to specify** the column names in SQL Query
- However, make sure the order of the values is in the same order as the columns in the table.
- Here, **syntax** would be as follows:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

Insert Statement

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

```
INSERT INTO Customers(customer_id, first_name,
last_name, age, country)
VALUES (5, 'Harry', 'Potter', 31, 'USA');
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Harry	Potter	31	USA

Insert Statement

Insert Row Providing Value Explicitly

- It's possible to provide default values to a column (for example, auto incrementing a column). In a database table, the **ID** field is usually unique auto incremented.
- In such cases, we can omit the value for that column during row insertion
- For example,

```
1  CREATE TABLE Customers(
2      ID int AUTO_INCREMENT,
3      FirstName varchar(25) NOT NULL,
4      LastName varchar(25),
5      Age int,
6      Country varchar(25)
7  );
```

```
INSERT INTO Customers(first_name, last_name, age, country)
VALUES
('James', 'Bond', 48, 'USA');
```

Insert Statement

Insert Multiple Rows at Once in SQL

- It's also possible to insert multiple rows to a database table at once.
- For example,

```
INSERT INTO Customers(first_name, last_name, age, country)
VALUES
('Harry', 'Potter', 31, 'USA'),
('Chris', 'Hemsworth', 43, 'USA'),
('Tom', 'Holland', 26, 'UK');
```

Insert Statement

Insert rows Without Specifying Column Names

- It is also possible to insert values in a row without specifying column names.
- For example

```
INSERT INTO Customers
VALUES
(5, 'Chris', 'Evans', 42, 'USA');
```

Insert Statement

Not Including All Columns During Insertion

- If we skip column names during row insertion, the values of those columns will be NULL

```
INSERT INTO Customers(first_name, last_name, age)
VALUES
('Brad', 'Pitt', 58);
```

Update Statement

- Update statement is used to modify the existing data present in the table
- To update data in table, SQL UPDATE command can be used
- To update all rows in table we can use following
- **Syntax:**

UPDATE <Table_name>

SET

column1=new_value

- For example,

```
UPDATE Customers
SET country = 'NP';
```

Update Statement

- The SQL UPDATE statement is used to edit existing rows in a database table. For example,

```
UPDATE Customers
SET first_name = 'Johnny'
WHERE customer_id = 1;
```

Update Multiple Values in a Row

- We can also update multiple values in a row at once. For example,

```
UPDATE Customers
SET first_name = 'Johnny', last_name = 'Depp'
WHERE customer_id = 1;
```

Update Statement

Update Multiple Rows

- The UPDATE statement can update multiple rows at once.
- For example,

```
UPDATE Customers
SET country = 'NP'
WHERE age = 22;
```

Delete Statement

- DELETE Statement is used to delete some or all records from existing table
- To delete data into a table, SQL DELETE command can be used

Delete all Rows in a Table

- The WHERE clause determines which rows to delete. However,
  we can delete all rows at once if we omit the WHERE clause.
- Syntax: DELETE FROM <TABLE_NAME>
- For example

```
DELETE FROM Customers;
```

Delete Statement

- In SQL, we use the DELETE statement to delete specific row(s) from a database table.
- **Syntax**:

DELETE FROM <TABLE_NAME> WHERE<Condition>

- For example,

```
DELETE FROM Customers
WHERE customer_id = 5;
```

SELECT Statement

- Select statement is used to retrieve data from database.
- SELECT query can never make any change in the database.
- The data returned by the SELECT query is in the form of result sets
- **Syntax**:

```
SELECT column1, column2, columnN FROM table_name;
```

- The SQL SELECT statement is used to select (retrieve) data from a database table. For example,

```
SELECT first_name, last_name
FROM Customers;
```

SELECT Statement

- To select all columns from a database table, we use the * character. For example,

```
SELECT *
FROM Customers;
```

- A SELECT statement can have an optional WHERE clause. The WHERE clause allows us to fetch records from a database table that matches specified condition(s). For example,

```
SELECT *
FROM Customers
WHERE last_name = 'Doe';
```

Data Control Language(DCL)

- DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.
- DCL is set of commands used to
  - **GRANT**- Gives user's privileges to perform task on database.
  - **REVOKE**-Withdraw user's privileges given by using the GRANT command.

Data Control Language(DCL)

Privileges:

- The set of actions that a user can perform on a database object are called the **privileges**
- Privilege is right to execute particular SQL statement on the database
- The high level user(DBA) has power to grant access to database and its objects
- Privileges can be of many types:
  - **System Privileges**: creating table
  - **Object Privileges**: execute query on table object
  - **Ownership Privileges**: execute query on tables created by same user

Data Control Language(DCL)

1. **System Privileges:**

- Rights and restriction implemented on database to control which user can access how much data in the database
- User requires system privileges to gain access to database
- System privileges are generally provided by DBA
- Few system privileges are as follows

System Privileges	Authorized to
CREATE USER	Create number of users in DBMS
DROP USER	Drop any other users in DBMS
CREATE ANY TABLE	Create table object in any schema
SELECT ANY TABLE	Query table object or view in any schema
DROP ANY TABLE	Drop table object in any schema

Data Control Language(DCL)

2. **Object privileges**

- Rights and restrictions to change contents of database objects.
- User requires object privileges to manipulate the content of object within database.
- Not all database users are allowed to make such changes in database; hence administrator should have control over all objects modification.
- The user which has GRANT ANY PRIVILEGE system privilege can act like administrator to control database modifications.
- Different objects has different privileges assigned for him.

Data Control Language(DCL)

- Few object privileges are as follows:

Object Privileges	Authorized To
SELECT	Select rows from table or view
INSERT	Add new rows to table or view
DELETE	Remove some rows from table or view
UPDATE	Modify content of rows from table or view
EXECUTE	To run procedure
REFERENCES	To reference a particular table using foreign key and check constraint

Data Control Language(DCL)

3. **Ownership privileges**

- Whenever you create a database object (like table or view) with the CREATE statement, you will become its owner and you will get full privileges for the table. (Like SELECT, INSERT, DELETE, UPDATE, and all other privileges).
- All other users are having no privileges on the newly created database object.
- You as owner of database object can explicitly give grant privileges to any other user by using the GRANT statement

Data Control Language(DCL)

Granting Privileges

- A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type.
- An authorized user may pass on this authorization to other users.

  This process is called as **granting of privileges**
- Generally GRANT statement is used by owner of table or view to give other users access permissions
- In SQL user accounts must be present in system before we can grant privileges to him.

Data Control Language(DCL)

Syntax:

GRANT <ALL | privilege list>
ON <Table_name or view_name>
TO <user| role list| PUBLIC>
[WITH GRANT OPTION]

[WITH GRANT OPTION]
It is used to allow user to grant privileges (which are granted to him) to other users.

Privilege List	Meaning
ALTER	Tables and Views
CREATE	Tables and Views
DROP	Tables and Views
DELETE	Tables and Views
INSERT	Tables and Views
SELECT	Tables and Views
UPDATE	Tables and Views
ALL	Tables and Views

Data Control Language(DCL)

Example

- Consider an example for granting update authorization to the Emp_Salary relation of the company database
- Assume that initially DBA grants update authorization on Emp_Salary to other users U1, U2, U3
- The following grant statement grants user U1, U2 and U3 the select privilege on Emp_Salary relation

GRANT SELECT, INSERT

ON mydb.*

TO mahesh'@'somehost;

Data Control Language(DCL)

- Following grant statement gives users all authorization on Emp_Salary relation using public keyword;

a) **Database privileges**

```
GRANT ALL
ON *.*
TO 'mahesh'@'somehost;
```

b) **Column privileges**

- This privilege authorizes a user to execute a function or procedure.

```
GRANT SELECT (col1), INSERT (col1, col2)
ON mydb.mytbl
TO 'mahesh'@'somehost'
```

Data Control Language(DCL)

c) **Table privileges**

This privilege authorizes a user to execute a function or procedure.

```
GRANT ALL
ON mydb.mytbl
TO 'mahesh'@'somehost';
```

OR

```
GRANT SELECT, INSERT
ON mydb.mytbl
TO 'mahesh'@'somehost';
```

Data Control Language(DCL)

Revoking Privileges

- We can reject privileges given to particular user with the help of revoke statement
- To revoke an authorization we use the revoke statement
- **Syntax:**

REVOKE <ALL | privileges list>
ON <relation name or view name>
FROM <user| role list | PUBLIC>
[RESTRICT/ CASCADE]

CASCADE: This will revoke all privileges along with all dependent grant privileges.

RESTRICT: This will not revoke all related grants only removes that GRANT only

Data Control Language(DCL)

Example:

- The revocation of privileges from user or role may cause other user or roles also have to leave that privilege. This behavior is called cascading of the revoke.

a) **To remove select privilege from users U1, U2 and U3**.

```
REVOKE SELECT
ON mydb.mytbl
FROM 'mahesh'@'somehost;
```

b) **To remove update rights on amount column of Emp_Salary from U1, U2 and U3.**

```
REVOKE UPDATE (amount)
ON EmpSalary
FROM 'mahesh'@'somehost
```

Data Control Language(DCL)

c) **To remove reference right on amount column from user U1.**

```
REVOKE REFERENCES (amount)
ON Emp_Salary
FROM 'mahesh'@'somehost;
```

The revoke statements may alternatively specify restrict if we don't want cascade behavior.

```
REVOKE SELECT
ON Emp_Salary
FROM 'mahesh'@'somehost
RESTRICT;
```

Difference Between DROP DELETE and TRUNCATE: DROP vs DELETE vs TRUNCATE

	DROP	DELETE	TRUNCATE
Definition	It completely removes the table from the database.	It removes one or more records from the table.	It removes all the rows from the existing table
Type of Command	It is a DDL command	It is a DML command	It is a DDL command
Syntax	DROP TABLE table_name;	DELETE FROM tble_nameWHERE conditions;	TRUNCATE TABLE table_name;
Memory Management	It completely removes the allocated space for the table from memory.	It doesn't free the allocated space of the table.	It doesn't free the allocated space of the table.
Effect on Table	Removes the entire table structure.	Doesn't affect the table structure	Doesn't affect the table structure
Speed and Performance	It is faster than DELETE but slower than TRUNCATE as it firstly deletes the rows and then the table from the database.	It is slower than the DROP and TRUNCATE commands as it deletes one row at a time based on the specified conditions.	It is faster than both the DELETE and DROP commands as it deletes all the records at a time without any condition.
Use with WHERE clause	Not applicable as it operates on the entire table	Can be used	It can't be used as it is applicable to the entire table

Transaction Control Language(TCL)

- Any SQL query can be executed with two basic operations on the database objects:
  - Read
  - Write
- After executing SQL query we must specify its final action as **commit** (save data) or **abort** (or revert back changes).
- The COMMIT statement ends the operations and makes all changes made to the data permanent on successful completion
- ABORT terminates and undoes all the actions done so far.

Transaction Control Language(TCL)

- TCL(Transaction Control Language) commands deals with the transaction within the database.
- Examples of TCL Commands:
  - **COMMIT**- Commits a Transaction
  - **ROLLBACK**- Rollbacks a transaction in case of any error occurs
  - **SAVEPOINT**- Sets a savepoint within a transaction

SAVEPOINT and ROLLBACK Command:

- SAVEPOINT is an indicator inside a transaction that is used for a partial rollback.
- When we are doing change to a transaction, we can create SAVEPOINTs to mark different points within the transaction.

Transaction Control Language(TCL)

- If at some stage we realize that an error is generated, then we can rollback up to a SAVEPOINT which we already created inside transaction.
- **Name of savepoint should be unique inside the transaction.**
- Suppose we create a savepoint having the same name as an previous savepoint, then the previous savepoint is deleted.
  - Once you have created savepoint after that you can perform other functions such as commit, roll back the entire transaction, or roll back to the savepoint.

Transaction Control Language(TCL)

Syntax for a SAVEPOINT command:

- **SAVEPOINT SAVEPOINT_NAME;**
- This command serves only in the creation of a SAVEPOINT among all the transactional statements.

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

```
SQL> SAVEPOINT SP1;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted.
SQL> SAVEPOINT SP2;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=2;
1 row deleted.
SQL> SAVEPOINT SP3;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=3;
1 row deleted.
```

Transaction Control Language(TCL)

mysql> **SAVEPOINT SP1;**

mysql> **SELECT * FROM CUSTOMERS;**

```
SQL> SELECT * FROM CUSTOMERS;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
6 rows selected.
```

TCL(Transaction Control Language)

ROLLBACK Command

- The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.
- This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.
- The **syntax** for a ROLLBACK command is as follows −

```
ROLLBACK TO SAVEPOINT_NAME ;
```

TCL(Transaction Control Language)

```
SQL> SAVEPOINT SP1;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted.
SQL> SAVEPOINT SP2;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=2;
1 row deleted.
SQL> SAVEPOINT SP3;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=3;
1 row deleted.
```

```
SQL> ROLLBACK TO SP2;
Rollback complete.
```

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

```
SQL> SELECT * FROM CUSTOMERS;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
6 rows selected.
```

TCL(Transaction Control Language)

COMMIT Command

- Changes made to the database by INSERT, UPDATE and DELETE commands are temporary until explicitly committed
- On execution of this command all changes to the database made by you are made permanent and cannot be undone
- **Syntax**: COMMIT [Work]

Exercise:1

- **For given database, write SQL queries:**

  EMPLOYEE(e_id,name,street,city)

  WORKS(eid,cid,salary)

  MANAGER(eid,manager_name)

  COMPANY(cid,company_name,city)

1. Modify database so that ‘Jack’ NOW LIVES IN ‘New York’
2. Give all employees of ‘ANZ corporation’ a 10% raise in salary

Solution: 1

1. mysql>
   UPDATE
   EMPLOYEE
   SET city= ‘New York’
   WHERE
   name=‘Jack’;

2. mysql>
   UPDATE
   WORKS
   SET salary = (salary+(0.1*salary))
   WHERE cid IN(SELECT cid FROM COMPANY

Exercise:2

- **For given database, write SQL queries:**

PERSON(driver_id#, name, address)

CAR(license, model, year)

ACCIDENT(report_no, date, location)

OWNS(driver_id#, license)

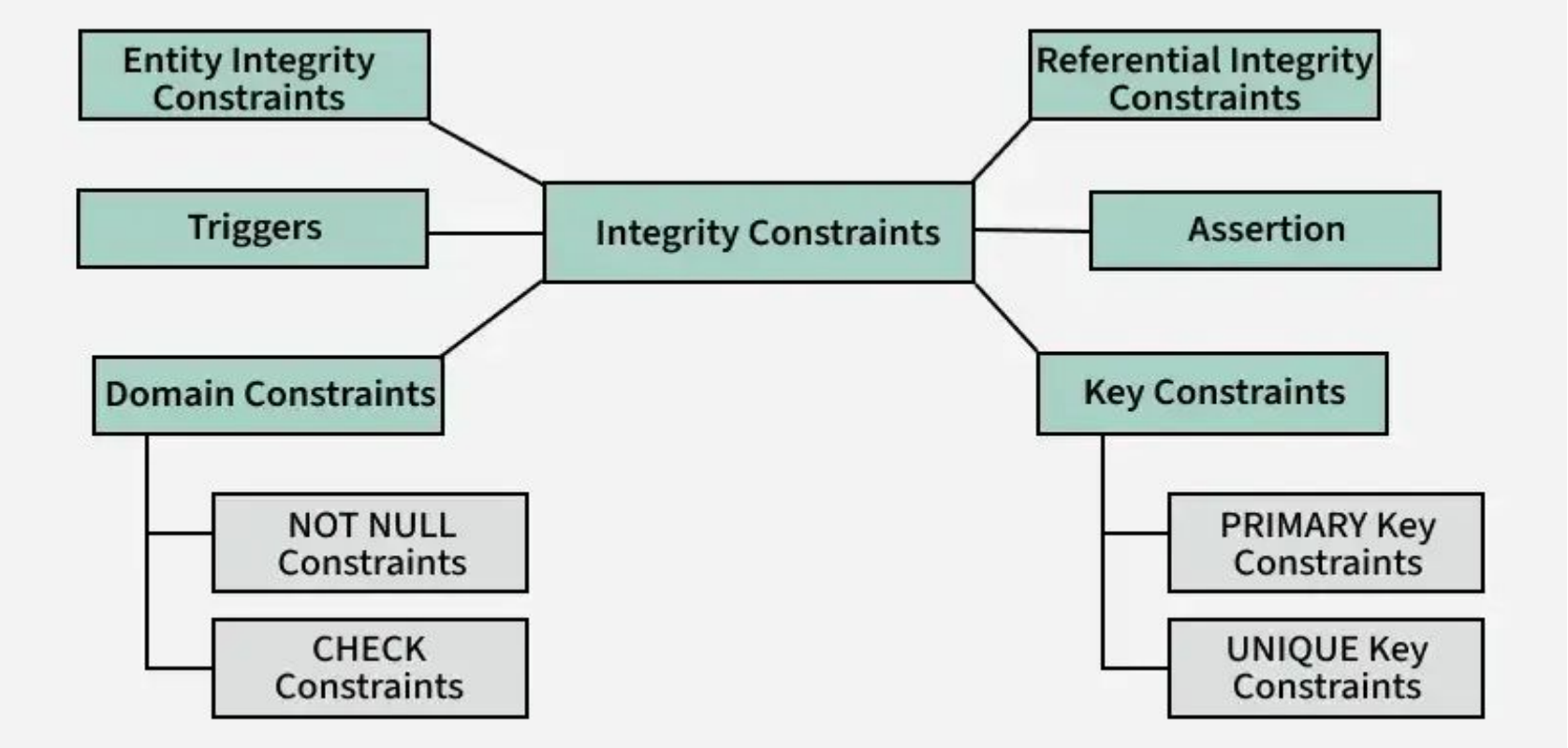
PARTICIPATED (driver_id, car, report_number, damage_amount)

1. Update damage amount for car with license number "Mum2022" in the accident with report number "AR2197" to Rs. 5000

Solution: 2

```
mysql>
UPDATE PARTICIPATED
SET damage_amount = 5000
WHERE report_number LIKE ‘AR2197’ AND car = ‘Mum2022’;
```

Integrity Constraints



Integrity Constraints

- Mainly **security and integrity** of a database is the most important factors in judging the success of system.
- Integrity constraint is a mechanism to prevent invalid data entry into table to maintain the data consistency.
- Constraints are used to enforce limits to the range of data or type of data that can be inserted/updated/deleted from a table
- The whole purpose of constraints is to maintain the data integrity during the various transactions like update/delete/insert on a table.

Types of Constraints

- There are different types of constraints:

1. Domain Integrity Constraints
2. Entity Integrity Constraints
3. Referential Integrity Constraints
4. Enterprise Constraints

Domain Integrity Constraints

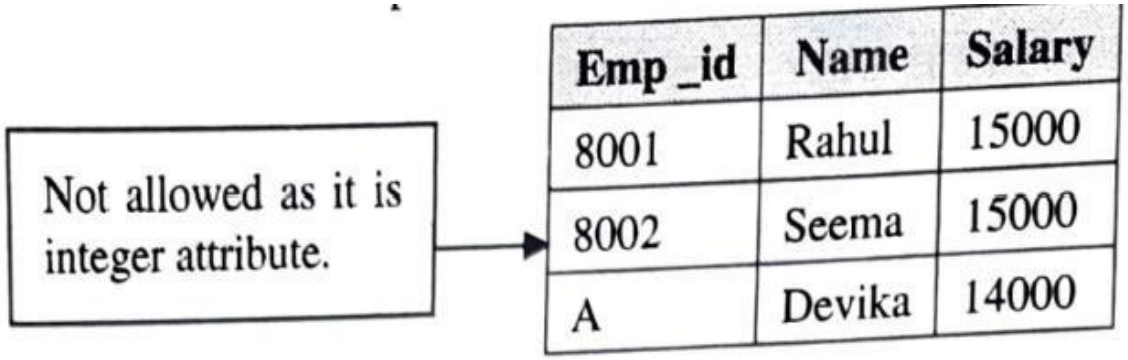
- The domain constraints are considered as the most basic form of integrity constraints.
- For attribute, domain integrity constraint defines the **default value, the range value or specific value.**
- The domain integrity constraints are easy to test when data is entered.
- The domain integrity constraints check that whether the attribute having proper and right value in the database or not.
- Domain integrity means it is the collection of valid set of values for an attribute.

Domain Integrity Constraints

- Constraints:
  - **Not Null**
  - **Unique**
  - **Default**
  - **Check**

Domain Integrity Constraints

- **Data Type**
  - A domain is the set of all unique values which are permitted for an attribute.
  - Domain constraints are user defined data type.
  - As we say that domain is the set of unique values, the column for which domain constraint has set, contains same type of data, based on its data type.
  - The column does not accept values of any other data type



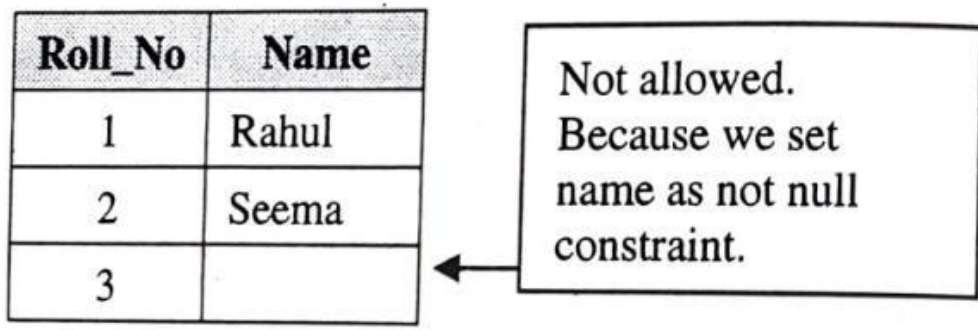
NOT NULL Constraints

- By setting the NOT NULL constraint we can assure that a column does not hold a NULL value
- When for a specific column, no value is provided while inserting a record into a table, by default it takes NULL values
- Example: Consider table student having 'name' field with NOT
  NULL constraint.
- **Syntax**:

CREATE TABLE Student
(Roll_No int NOT NULL,

Name varchar(10) NOT

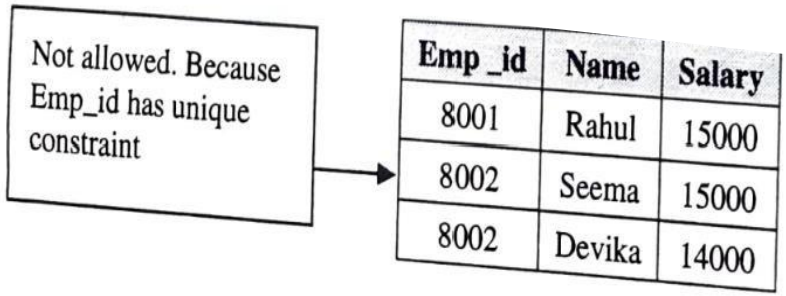
NULL



Unique Constraint

- UNIQUE Constraint as the name suggests, it can take only unique values in a column or set of columns
- It keeps uniqueness of the table.
- When a column has a unique constraint then that particular column cannot have duplicate values in it.
- **Syntax**:

```
CREATE TABLE Employee
(Emp_id int UNIQUE,
Name varchar(10),
Salary Double(10,2)
);
```



Default Constraint

- When a user does not provide a value to the column while inserting the records in the table, the DEFAULT constraint provides a default value to that column.
- Example: We can set DEFAULT Constraint by assigning the value 10000 to the column exam_fees in student table
- **Syntax**:

```
CREATE TABLE Student(
Roll_No int NOT NULL,
Name varchar (25),
Fees int DEFAULT 10000
);
```

Check Constraint

- This constraint is used to set user defined constraint for the column
- As per the requirements of business for which we are developing the application, we may have to set some rules while inserting or updating data on specific field/ attribute
- **Syntax:**

```
CREATE TABLE Student(
Roll_No int UNIQUE,
Name varchar(25),
Age int,
CHECK Age between 15 and 20);
```

Entity Integrity Constraints

Key Constraint – Primary Key

- Under Entity Integrity Constraint Primary key is the main factor
- Primary key uniquely identify each record in a table.
- It must have unique values and cannot hold null values
- Primary key is the combination of NOT NULL & UNIQUE constraints.

Not allowed as primary key cannot be null

Emp_id	Name	Salary
8001	Rahul	15000
8002	Seema	15000
	Devika	14000

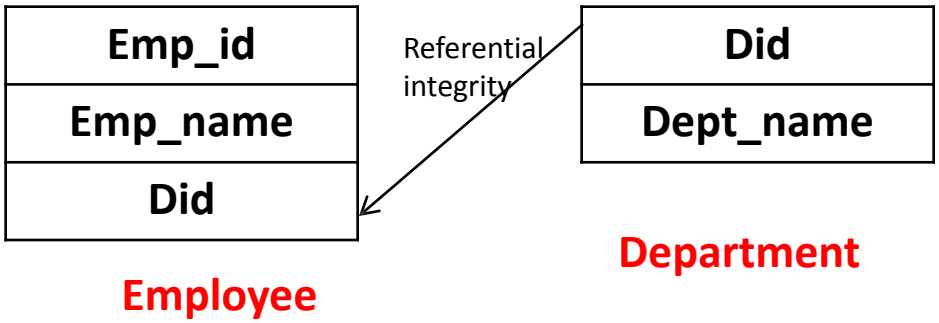
Entity Integrity Constraints

- Here we set primary key to emp_id table. If we add any repeated value or null value in the column it will because primary key never contains null or repeated values.
- **Syntax:**

```
CREATE TABLE Employee(

Emp_id int,

Name varchar(25),

Salary char(4),

PRIMARY KEY (Emp_id)

)
```

Referential Integrity Constraints

- A value appearing in a one relation (table) for a given set of attributes also appears for another set of attributes in another relation (table). This is called **referential integrity.**
- The referential integrity constraint is specified between two tables to maintain the consistency among tuples in the two tables.
- The tuple in one relation refers only to an existing tuple in another relation.



Referential Integrity Constraints

Emp_id	Emp_name	Did
1	Sam	20
2	Suhas	10
3	Jay	20
4	Om	10

Did	Dept_name
10	HR
20	TIS
30	L&D

- Ex: Employee table has Did as foreign key reference to **Did** column in Department table this is called as referential integrity.
- Here we are forcing the database to check the value of Did column from the department table while inserting any value in Employee table.
- This helps to maintain data consistency

Referential Integrity Constraints

Foreign key violations in SQL

- If any row in EMP table is added with "Did" value which is not there in department table the insert statement will give foreign key violation error.
- In previous example, we will refer Department as parent table (as it is containing Primary key) and Employee table as Child table (as it is containing Foreign Key).
- There are 4 problems which causes the foreign key violations:

A. **Adding new tuple to Child Table (Add Child)**
B. **Updating tuple from Child Table**
C. **Deleting tuple from Parent Table**
D. **Updating tuple from Parent Table**

Referential Integrity Constraints

a) Adding new tuple to Child Table (Add Child)

- If we try to add an employee with Did 70 to employee table(child table) , it will return foreign key violation error
- As Did 70 is not there in Department table(Parent table)

Example:

- INSERT INTO Employee VALUES (11,'Devid', 70);

Output

- ORA-02291: Integrity constraint (Employee.FK_Employee) violated - parent key not found

Emp_Id	Emp_name	Did
11	Devid	70

- This functionality helps to maintain data consistency in database.

Referential Integrity Constraints

b) Updating tuple from Child Table

- If we try to update an employee Emp_id = 2 with Did as 70 to employee table (Child Table), it will return foreign key violation error.
- As Did 70 is not there in Department table (Parent table)
- **Example**:

UPDATE Employee

SET Did= 70 WHERE Emp_id=2;

Output:

ORA-02291 Integrity constraint (Employee.FK_Employee) violated - parent key not found

- This functionality helps to maintain data consistency in database.

Referential Integrity Constraints

c) Deleting tuple from Parent Table

- If we try to delete department Did = 10 from Department table (Parent table), it will return foreign violation error.
- As there are few employees working in department with Did =10.
- **Example:**

DELETE Department WHERE Did=10;

- *Output:*

ORA-02292: integrity constraint(Employee.FK_Employee) violated - child record found.

- This functionality will creates limitation for deletion of parent record if it has some associated child records

Referential Integrity Constraints

d) Updating tuple from Parent Table

- If we try to update department of Did = 10 with Did = 70, it will return foreign key violation are few employees still working in department with Did = 10.
- **Example:**

```
UPDATE Department SET Did = 70 WHERE Did = 10;
```

Output:

ORA-02292: integrity constraint (Employee.FK_Employee) violated - child record found.

- This functionality will creates limitation for updating parent
  record if it has some associated child records

Referential Integrity Constraints

Delete-Update (DU) rules to solve problem of foreign key violation

- If any row in EMP table is added with 'Did' value which is not there in department table then insert statement will give foreign key violation error
- This rule can be enforced as given as

  follows: Create Table Employee(

Eid varchar (50) Primary Key,

…

Did varchar (50) foreign key references department

(Did) On delete CASCADE

On update CASCADE);

Referential Integrity Constraints

NO ACTION / RESTRICT

- This clause will discards the delete or update operation on the parent table
- In this case the database engine will not allow user to delete the row and using FK violation error.
- The RESTRICT rule will not allow you to delete a row from the parent table although as there corresponding row present in child table.

```
Create Table Employee(
Eid varchar (50) Primary Key,
…
Did varchar (50) foreign key references
department (Did)
On delete RESTRICT
On update RESTRICT) ;
```

Referential Integrity Constraints

- The database engine will give the error and the delete action on the row in the parent table is ignored
- Deletion of department is not allowed as there is some employees are present in that department

CASCADE

- Corresponding rows are deleted from the referencing table (Child table), if that row is deleted from parent table.
- If a department is deleted then all the employee records that refers to the deleted department are also been deleted

```
Create Table Employee(
Eid varchar (50) Primary Key,
…
Did varchar (50) foreign key references
department (Did)
On delete CASCADE
On update CACADE) ;
```

Referential Integrity Constraints

SET NULL

- Foreign Key data value is set to NULL, if the corresponding row in the parent table is deleted
- For this constraint to execute, the foreign key columns must be nullable
- Insert Null value of did in the place of deleted did in employee table.

```
Create Table Employee(
Eid varchar (50) Primary Key,
…
Did varchar (50) foreign key references department
(Did)
On delete SET NULL
On update SET NULL) ;
```

Referential Integrity Constraints

SET DEFAULT

- Foreign key data values refer to non-existing foreign key are set to their default values.
- For this constraint to execute, all foreign key columns must have default definitions.
- If a column is null able, and there is no explicit default value set, NULL becomes the implicit default value of the column.
- Insert any default value of 'did' (which exists in the departing table) in the place of deleted 'Did'

```
Create Table Employee(
Eid varchar (50) Primary Key,
…
Did varchar (50) foreign key references department
(Did)
On delete SET DEFAULT
On update SET DEFAULT) ;
```

Difference: Primary key and Foreign Key Constraint

Parameter	Primary Key	Foreign Key
Function	Primary key uniquely identify a record in the table.	Foreign key is a field in the table that is primary key in another table.
Null	Primary Key can't accept null values.	Foreign key can accept null values.
Index	By default, Primary key is clustered index and data in Foreign key do not automatically create an index, the database table is physically organized in the clustered or non-clustered. You can manually sequence of clustered index.	Foreign key do not automatically create an index, clustered or non-clustered. You can manually create an index on foreign key
Number	Only one primary key in a table	More than one foreign key in a table

Referential Integrity Constraints

Enterprise Constraints

- Enterprise Constraints are also referred as Semantic constraints.
- They are additional rules specified by users or database administrators.
- These **rules are depending upon the requirements and constraints of the business** for which the database system is being maintained.
- For Example:
  - In College System a class can have a maximum of 30 students. A teacher can teach a maximum of 4 classes a semester.
  - In Corporate System an employee cannot take a part in more than 5 projects. Salary of an employee cannot exceed salary of the employee's manager

Thank You!!

Module 4:
Structured Query Language (SQL)

4.2 Outline:

- Set and string operations
- Aggregate function-group by, having clause
- Views in SQL
- Joins, Nested and Complex Queries
- Triggers

SET Operations

- Set operations are supported by SQL to be performed on table data using **Set Operators**
- In order to execute set operations, two queries must be “set compatible”
  - Both relations have same number of columns
  - Corresponding column are data type compatible
- Queries which contain set operators are called **compound queries.**
- The different Set Operators are as follows:
  - Union
  - Union All
  - Intersect
  - Minus (Except)

Union Operator

- UNION combines the results of two or more SELECT statements.
- After performing the UNION operation, the **duplicate rows will be eliminated from the results.**

- **Syntax:**

```
SELECT expression_1, expression_2, ... , expression_n FROM table_1
UNION
SELECT expression_1, expression_2, ... , expression_n FROM table_2
```

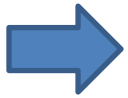
Union Operator

Customer

first_name	last_name
Stephen	Jones
Mark	Smith
Denise	King
Paula	Johnson
Richard	Archer

Employee

first_name	last_name
Christina	Jones
Michael	McDonald
Paula	Johnson
Stephen	Jones
Richard	Smith



```
SELECT first_name, last_name
FROM customer
UNION
SELECT first_name, last_name
FROM employee;
```

Result:

first_name	last_name
Stephen	Jones
Mark	Smith
Denise	King
Paula	Johnson
Richard	Archer
Christina	Jones
Michael	McDonald
Richard	Smith

Union ALL Operator

- UNION ALL is also used to combine the results of two or more SELECT statements.
- After performing the UNION ALL operation, the **duplicate rows will not be eliminated** from the results, and all the data is displayed in the result without removing the duplication.

- **Syntax**:

```
SELECT expression_1, expression_2, ... , expression_n FROM table_1
                              UNION ALL
SELECT expression_1, expression_2, ... , expression_n FROM table_2
```

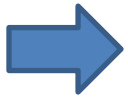
Union ALL Operator

Customer

first_name	last_name
Stephen	Jones
Mark	Smith
Denise	King
Paula	Johnson
Richard	Archer

Employee

first_name	last_name
Christina	Jones
Michael	McDonald
Paula	Johnson
Stephen	Jones
Richard	Smith



```
SELECT first_name, last_name
FROM customer
UNION ALL
SELECT first_name, last_name
FROM employee;
```

Result:

first_name	last_name
Stephen	Jones
Mark	Smith
Denise	King
Paula	Johnson
Richard	Archer
Christina	Jones
Michael	McDonald
Paula	Johnson
Stephen	Jones
Richard	Smith

Intersect Operator

- The INTERSECT operator allows you to find the results that exist in both queries.
- After performing the INTERSECT operation, the data/records which are common in both the SELECT statements are returned.

- **Syntax**:

```
SELECT expression_1, expression_2, ... , expression_n FROM table_1
INTERSECT
SELECT expression_1, expression_2, ... , expression_n FROM table_2
```

Intersect Operator

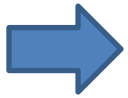
Customer

first_name	last_name
Stephen	Jones
Mark	Smith
Denise	King
Paula	Johnson
Richard	Archer

Employee

first_name	last_name
Christina	Jones
Michael	McDonald
Paula	Johnson
Stephen	Jones
Richard	Smith

```
SELECT first_name, last_name
FROM customer
INTERSECT
SELECT first_name, last_name
FROM employee;
```



Result:

first_name	last_name
Stephen	Jones
Paula	Johnson

Minus/ Except Operator

- The MINUS operator allows you to filter out the results which are **present in the first query but absent in the second query.**
- After performing the MINUS operation, the data/records which are not present in the second SELECT statement or query are displayed.
- **Syntax:**

```
SELECT expression_1, expression_2, ... , expression_n FROM table_1
                              EXCEPT
SELECT expression_1, expression_2, ... , expression_n FROM table_2
```

Note:

- The MINUS operator is supported only in Oracle databases.
- For other databases like SQLite, PostgreSQL, SQL server, you can use **EXCEPT** operator to perform similar operations.

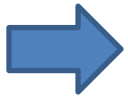
Minus Operator

Customer

first_name	last_name
Stephen	Jones
Mark	Smith
Denise	King
Paula	Johnson
Richard	Archer

Employee

first_name	last_name
Christina	Jones
Michael	McDonald
Paula	Johnson
Stephen	Jones
Richard	Smith



Our query would look like this:

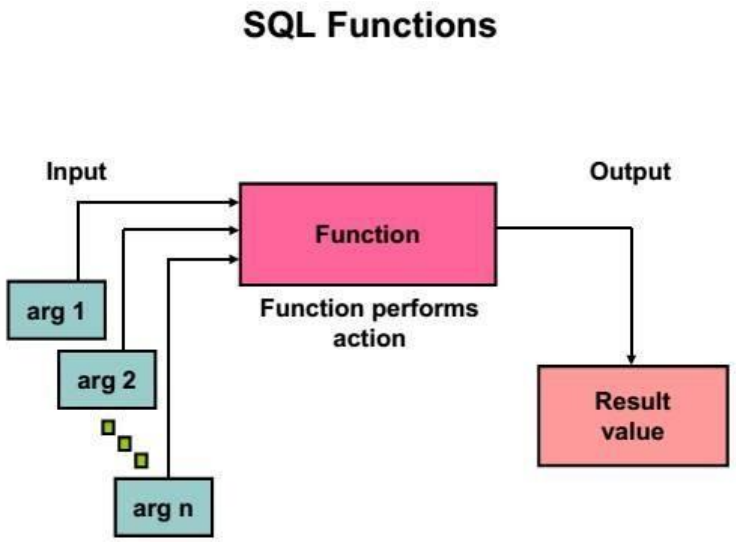
```
SELECT first_name, last_name
FROM customer
EXCEPT
SELECT first_name, last_name
FROM employee;
```

Result:

first_name	last_name
Mark	Smith
Denise	King
Richard	Archer

Built In Functions

- Functions accepts multiple input arguments and after processing it returns only one result value
- Types of Functions:
  - **MATH (NUMERIC)**
  - **STRING**
  - **DATE**



Built In Functions: MATH

- Functions applied to columns with data type number or numeric type

Function	Description
ABS	Returns the absolute value of a number.
AVG	Returns the average value of an expression/column values.
CEILING	Returns the nearest integer value which is larger than or equal to the specified decimal value.
COUNT	Returns the number of records in the SELECT query.
FLOOR	Returns the largest integer value that is less than or equal to a number. The return value is of the same data type as the input parameter.
MAX	Returns the maximum value in an expression.
MIN	Returns the minimum value in an expression.
RAND	Returns a random floating point value using an optional seed value.
ROUND	Returns a numeric expression rounded to a specified number of places right of the decimal point.
SIGN	Returns an indicator of the sign of the input integer expression.
SUM	Returns the sum of all the values or only the distinct values, in the expression. NULL values are ignored.

Built In Functions:
MATH

- **ABS()**: Return the absolute value of a number

```
SELECT Abs(-243.5) AS AbsNum;
```

Result:

Number of Records: 1

AbsNum
243.5

- **POWER(a,b):** The POWER() function returns the value of a number raised to the power of another number.

```
SELECT POWER(8, 3);
```

Result:

Number of Records: 1

512

Built In Functions:

MATH

- **ROUND()**: Return the round value to specified decimal.
  - Increasing value of previous digit by 1 if last digit is equal to or above 5
  - If last digit is below 5, don't change value of previous digit

Syntax: SELECT ROUND(45.926,2)

Output: 45.93

- **MOD(N,M) or N%M:** Function returns the remainder of N divided by number M

Syntax: SELECT MOD(4589,100)

Output: 89

Built In Functions:

MATH

- **CEIL():** It returns the smallest integer value that is greater than or equal to a number.

Syntax: SELECT CEIL(25.75)

Output: 26

- **FLOOR():** It returns the largest integer value that is less than or equal to a number.

Syntax: SELECT FLOOR(25.75)

Output: 25

Built In Functions:

MATH

- **GREATEST(exp1, exp2, exp3, exp_n):** It returns the greatest value in a list of expressions.

Syntax: SELECT GREATEST(30, 2, 36, 81, 125)

Output: 125

- **LEAST(exp1, exp2, exp3, exp_n):** It returns the smallest value in a list of expressions.

Syntax: SELECT LEAST(30, 2, 36, 81, 125)

Output: 2

- **SQRT():** It returns the square root of a number.

Syntax: SELECT SQRT(25)

Output: 5

Built In Functions: STRING

- MySQL string functions are used to manipulate string data and derive some information and analysis from tables
- These functions can be applied to column having data type CHAR, VARCHAR

String Function name		
ASCII()	CHARINDEX()	LEN()
CHAR()	LEFT()	LOWER()
NCHAR()	RIGHT()	RIGHT()
LOWER()	UPPER()	LTRIM()
RTRIM()	REPLACE()	REPLICATE()
REVERSE()	SPACE()	STUFF()
UNICODE()	QUOTENAME()	FORMAT()
CONCAT()		

Built In Functions: STRING

- **LOWER(String):** The LOWER() function converts a string to lower-case.

```
SELECT LOWER('SQL Tutorial is FUN!');
```

Result:

Number of Records: 1

sql tutorial is fun!

- **UPPER(String):** The UPPER() function converts a string to upper-case.

```
SELECT UPPER('SQL Tutorial is FUN!');
```

Result:

Number of Records: 1

SQL TUTORIAL IS FUN!

Built In Functions: STRING

- **LPAD(char1, n, char2):** This function is used to make the given string of the given size by adding the given symbol

Syntax: LPAD('geeks', 8, '0')

Output: 000geeks

- **LTRIM(string, chars):** This function is used to cut the given sub string from the original string

Syntax: LTRIM('123123geeks', '123');

Output: geeks

Built In Functions: STRING

- **RPAD(char1,n, cahr2):** This function is used to make the given string as long as the given size by adding the given symbol on the right

Syntax: RPAD('geeks', 8, '0');

Output: ‘geeks000’

- **RTRIM(string, char):** This function is used to cut the given sub string from the original string.

Syntax: RTRIM('geeksxyz', 'xyz');

Output: ‘geeks’

Built In Functions: STRING

- **TRIM():** This function is used to cut the given symbol from the string

Syntax: TRIM(LEADING '0' FROM '000123');

Output: 123

- **REPLACE(string, old_string, new_string) :** This function is used to cut the given string by removing the given sub string.

Syntax: SELECT REPLACE('SQL Tutorial', 'SQL', 'HTML');

Output: HTML Tutorial

Built In Functions: STRING

- **LENGTH(string):** This function is used to find the length of a word

Syntax: LENGTH('GeeksForGeeks');

Output: 13

- **SUBSTR(string,m,n):** This function is used to find a sub string from the a string from the given position

Syntax:SUBSTR('geeksforgeeks', 1, 5);

Output: ‘geeks’

Built In Functions:

DATE

- These functions can be applied to columns with data type Date and Time
- Date time functions can be applied in column having date and time data type

NOW(): This function will return today's date and time

- **Syntax:** SELECT NOW();
- **Output**: Today 2023-02-13 05:25:51

Function	Result
YEAR(NOW())	2023
HOUR(NOW())	5
MIN(NOW())	25
SEC(NOW())	51

Aggregate Functions

- For decision making, we need to summarize data from table like average, sum, minimum etc.
- SQL provides aggregate functions which can summarize data of given table by performing a calculation on set of values and return a single value
- Usually these functions ignore NULL values(except for COUNT)
- Different types of Aggregate Functions are:
  - Min(C): returns minimum value in column C
  - Max (C) : returns maximum value in column C
  - Sum(C) : sum of all values in column C
  - Avg(C) : average of all values in column C
  - Count(C) : number of values in column C

Sample Table

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Aggregate Functions: Min( )

- **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:

```
SELECT MIN(RATE)
FROM PRODUCT_MAST;
```

Output:

```
10
```

Sample Table

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Aggregate Functions: Max( )

- **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:

```
SELECT MAX(RATE)
FROM PRODUCT_MAST;
```

```
30
```

Sample Table

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Aggregate Functions: Sum( )

- **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: SUM()

```
SELECT SUM(COST)
FROM PRODUCT_MAST;
```

```
670
```

Aggregate Functions: Sum( )

Example: SUM() with WHERE

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3;
```

Output:

```
320
```

Example: SUM() with GROUP BY

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3
GROUP BY COMPANY;
```

Output:

```
Com1    150
Com2    170
```

Sample Table

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Aggregate Functions: Avg( )

- 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:

```
SELECT AVG(COST)
FROM PRODUCT_MAST;
```

Output:

```
67.00
```

Sample

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Aggregate Functions: Count( )

- 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.**

Example: COUNT()

```
SELECT COUNT(*)
FROM PRODUCT_MAST;
```

Output:

```
10
```

Sample

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Aggregate Functions: Count( )

Example: COUNT with WHERE

```
SELECT COUNT(*)
FROM PRODUCT_MAST;
WHERE RATE>=20;
```

Output:

```
7
```

Example: COUNT() with DISTINCT

```
SELECT COUNT(DISTINCT COMPANY)
FROM PRODUCT_MAST;
```

Output:

```
3
```

Sample

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Aggregate Functions: Count( )

Example: COUNT() with GROUP BY

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY;
```

Output:

```
Com1    5
Com2    3
Com3    2
```

Order By Clause

- The ORDER BY keyword is used to sort the result-set by a specified column.
- The ORDER BY keyword sorts the records in ascending order by default.
- If you want to sort the records in a descending order, you can use the DESC keyword.
- **Syntax**:

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

Order By Clause

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

```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY NAME DESC;
```

This would produce the following result −

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

Order By Clause

```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY NAME, SALARY;
```

This would produce the following result −

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

Group By Clause

- Related rows can be grouped together by **GROUP BY clause** based on distinct values that exist for specified columns.
- A GROUP BY clause creates a set of data, containing several sets of records grouped together based on some condition.
- The GROUP BY statement is used with the SQL aggregate functions to group the retrieved data by one more columns or expression.
- GROUP BY column have to be in the SELECT clause
- Rows with same values for grouping columns are placed in distinct groups. Each group is treated as single row in query result

Group By Clause

- This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- **Syntax**:

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
```

Group By Clause

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

```
SQL> SELECT NAME,
SUM(SALARY)
FROM CUSTOMERS
GROUP BY NAME;
```

NAME	SUM(SALARY)
Hardik	8500.00
kaushik	8500.00
Komal	4500.00
Muffy	10000.00
Ramesh	3500.00

Having Clause

- The **HAVING Clause** enables you to specify conditions that filter which group results appear in the results.
- The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.
- A query can contain both WHERE and HAVING clause
- **Syntax**:

```
SELECT
FROM
WHERE
GROUP BY
HAVING
```

Having Clause

Table: Sales

SaleID	Product	Quantity	Price
1	Laptop	2	50000
2	Mobile	5	20000
3	Laptop	3	50000
4	Tablet	4	15000
5	Mobile	6	20000
6	Laptop	1	50000

```
SELECT Product, SUM(Quantity) AS Total_Quantity
FROM Sales
GROUP BY Product
HAVING SUM(Quantity) > 5;
```

Output:

Product	Total_Quantity
Laptop	6
Mobile	11

Views in SQL.

- In SQL, a view is a **virtual table** containing the records of one or more tables based on SQL statement executed.
- Just like a real table, view contains rows and columns.
- You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table
- The changes made in the table get automatically reflected in original table and vice versa

Views in SQL

Purpose of View:

- View is very useful in maintaining the security of database
- Consider a base table employee having following data

Emp_id	Emp_name	Salary	Address
E1	Kunal	8000	Camp
E2	Jay	7000	Tilak Road
E3	Radha	9000	Somwar Peth
E4	Sagar	7800	Warje
E5	Supriya	6700	LS Road

Views in SQL

1. Now just consider we want to give this table to any user but **don't want to show him salaries** of all the employees

In that we can create view from this table which will contain only the part of base table which we wish to show to user

Emp_id	Emp_name	Address
E1	Kunal	Camp
E2	Jay	Tilak Road
E3	Radha	Somwar Peth
E4	Sagar	Warje
E5	Supriya	LS Road

Views in SQL

2. Also in multiuser system, it may be possible that more than one user may want to update the data of some table

   - Consider two users A and B want to update the employee table.

     In such case we can give views to both these users

   - These users will make changes in their respective views, and their respective changes are done in the base table automatically

1. Creating View

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

- A view can be created using the **CREATE VIEW** statement.
- We can create a view from a single table or multiple tables.

1. Creating

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

1. Creating View

Creating View from multiple tables

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

1. Creating

Query:

```
CREATE VIEW MarksView AS
SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
FROM Student_Detail, Student_Mark
WHERE Student_Detail.NAME = Student_Marks.NAME;
```

```
SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

2. Updating View

- Update query is used to update the records of view
- Updation in view reflects the original table also means same changes will be made in the original table

```
UPDATE < view_name > SET<column1>=<value1>,<column2>=<value2>,...
WHERE <condition>;
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO
A007	Ramasundar	Bangalore	0.15	077-25814763
A003	Alex	London	0.18	075-12458969
A008	Alford	New York	0.12	044-25874365
A011	Ravi Kumar	Bangalore	0.15	077-45625874
A010	Santakumar	Chennai	0.14	007-22388644
A012	Lucida	San Jose	0.12	044-52981425

2. Updating View

```
UPDATE agentview
SET commission=.13
WHERE working_area='London';
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO
A007	Ramasundar	Bangalore	0.15	077-25814763
A003	Alex	London	0.13	075-12458969
A008	Alford	New York	0.12	044-25874365
A011	Ravi Kumar	Bangalore	0.15	077-45625874
A010	Santakumar	Chennai	0.14	007-22388644
A012	Lucida	San Jose	0.12	044-52981425

2. Updating View

- In case of view containing joins between multiple tables, only insertion and updation in the view is allowed, deletion is not allowed
- Data modification is **not allowed** in the view which is based on union queries
- Data modification is **not allowed** in the view where GROUPBY or DISTINCT statements are used
- In view, text and image columns **can’t be modified**

3. Dropping View

- Drop query is used to delete a view

Syntax

```
DROP VIEW view_name;
```

- **Example**:

```
DROP VIEW MarksView;
```

Types of Views

- There are **two** types of Views:

1. **Simple View**

- The views which are based on **only one table** called as Simple view.
- Allow to perform DML (Data Manipulation Language) operations with some restrictions.
- Query defining simple view **cannot have** any join or grouping condition.

Types of Views

2. Complex View

- The views which are based on **more than one table** called as complex view.
- Do not allow DML operations to be performed.
- Query defining complex view can have join or grouping condition.

Difference: Simple and Complex View

Simple View	Complex View
Contains only one single base table or is created from only one table.	Contains more than one base table or is created from more than one table.
We cannot use group functions like MAX(), COUNT(), etc.	We can use group functions.
INSERT, DELETE and UPDATE are directly possible on a simple view.	We cannot apply INSERT, DELETE and UPDATE on complex view directly.
Example: CREATE VIEW Employee AS SELECT Empid, Empname FROM Employee WHERE Empid = '030314';	**Example:** CREATE VIEW EmployeeByDepartment AS SELECT e.emp_id, d.dept_id, e.emp_name FROM Employee e, Department d WHERE e.dept_id=d.dept_id;

Views : Advantages

1. **Security**: restrict user from accessing all data
2. **Hides Complexity**: view can be generated from complex query so result can be stored instead of writing complicated query again and again
3. **Dynamic Nature:** view definition remains unaffected if any changes are made in base table except table drop of column is altered
4. **Does not allows direct access to tables of data dictionary:** data dictionary cannot be damaged or changed

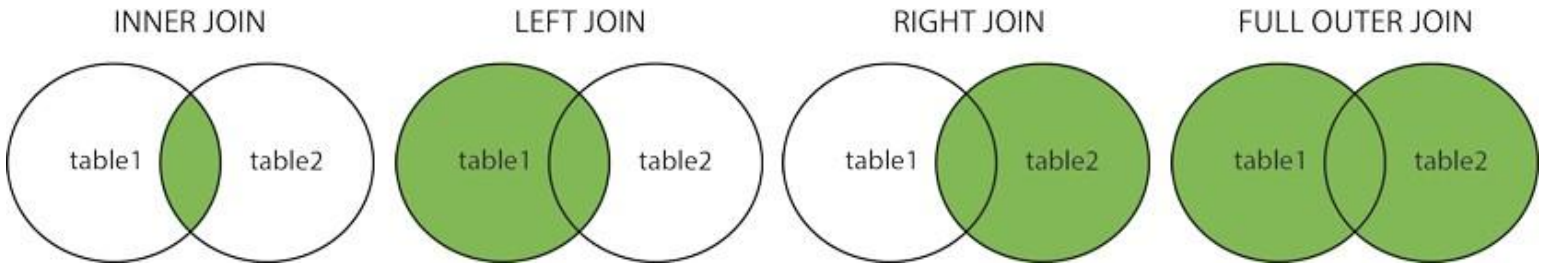
Views : Limitations

1. **Performance**: If view is defined by complex multi-table query user are not aware of how much complicated task query is performing
2. **View Management**: Views should be created as per standards so that job of DBA is simplified otherwise it becomes difficult to manage views
3. **Update Restrictions:** Whenever a user tries to update view, DBMS must translate query and apply updates on rows of underlying base table

Join

- A join operation means combining columns from one(self-table) or more tables by using values which are common
- There are six types of Joins:

1. INNER
2. LEFT OUTER
3. RIGHT OUTER
4. FULL OUTER
5. SELF

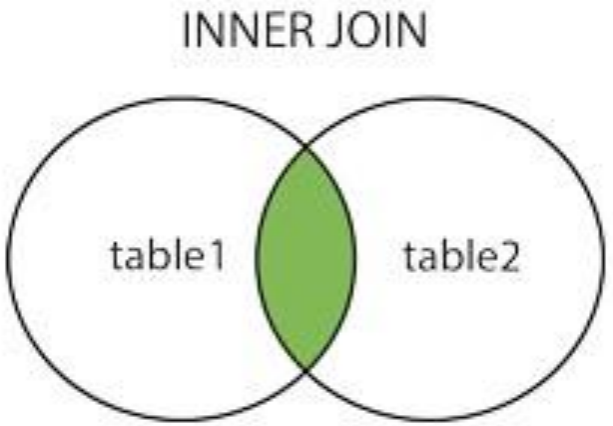


1. INNER JOIN

- The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```



1. INNER JOIN

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Student

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

StudentCourse

1. INNER JOIN

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

Output

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

Output

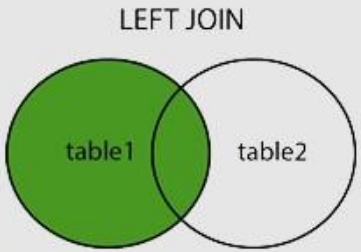
2. LEFT JOIN

- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2).

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.



2. LEFT JOIN

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Student

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

StudentCourse

2. LEFT JOIN

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

Output

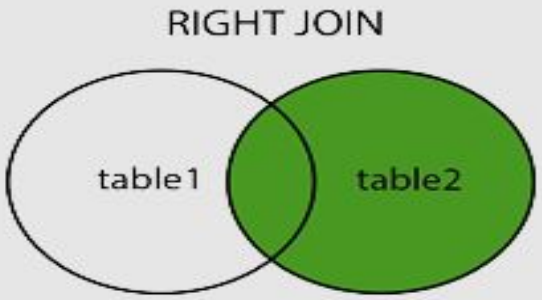
3. RIGHT JOIN

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1).

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases `RIGHT JOIN` is called `RIGHT OUTER JOIN`.



3. RIGHT JOIN

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Student

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

StudentCourse

3. RIGHT JOIN

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

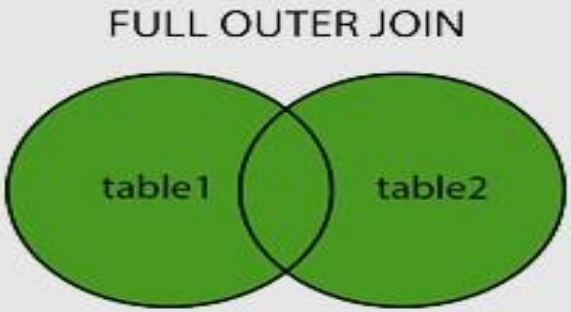
Output

4. FULL OUTER JOIN

- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
- **Tip:** FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



4. FULL OUTER JOIN

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Student

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

StudentCourse

4. FULL OUTER JOIN

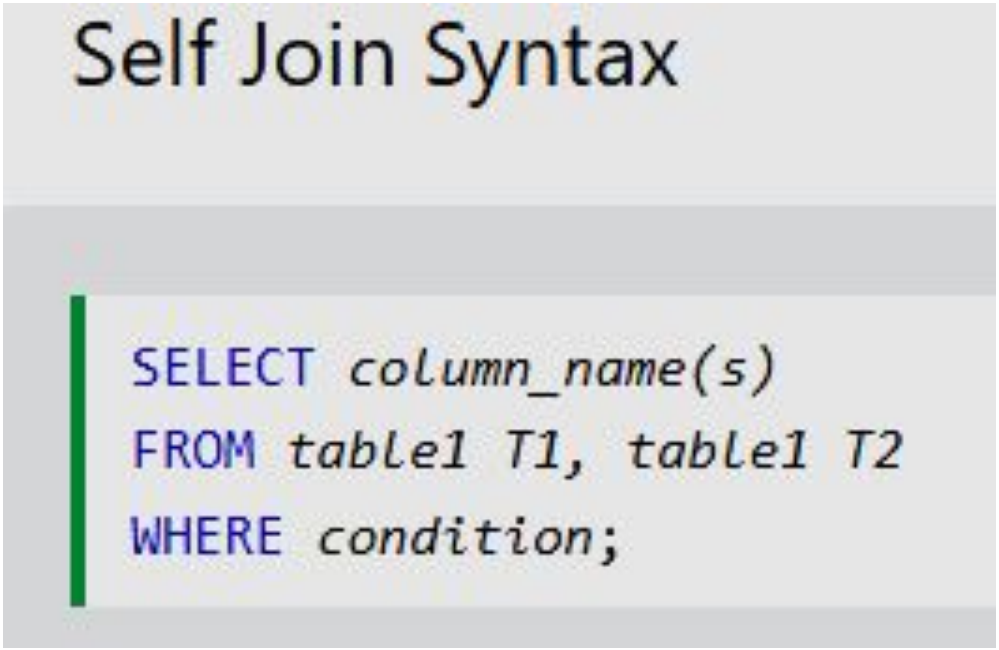
```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL
NULL	4
NULL	5
NULL	4

5. Self Join

- A self join is a regular join, but the table is joined with itself.



5. Self Join

Id	Name	Age	City	Salary
1	Rohan	30	Mumbai	50000
2	Mohan	35	Pune	30000
3	Kriti	40	Mohali	75000
4	Riran	25	Patana	35000
5	Meenal	40	Mumbai	750000
5	Meenal	40	Mumbai	750000

Project_No	Id	Department
101	1	Testing
102	2	Development
103	3	Designing
104	4	Development

```
SELECT e.employee_name AS employee,
m.employee_name AS manager
FROM GFGemployees AS e
JOIN GFGemployees AS m
ON e.manager_id = m.employee_id;
```

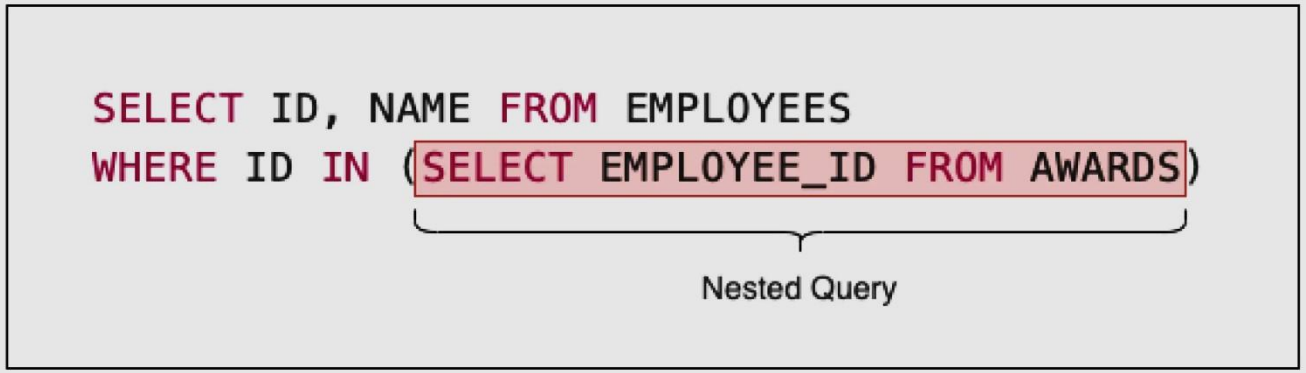
employee	manager
Zaid	Raman
Rahul	Raman
Raman	Kamran
Farhan	Kamran

Nested and Complex Queries

- Nested query is one of the most useful functionalities of SQL.
- Nested queries are useful when we want to write complex queries where **one query uses the result from another query**.
- Nested queries will have multiple SELECT statements nested together.
- A SELECT statement nested within another SELECT statement is called a **subquery.**

Nested Query

- A nested query in SQL contains a query inside another query.
- The result of the inner query will be used by the outer query.
- For instance, a nested query can have two **SELECT** statements, one on the inner query and the other on the



Types of Nested Queries

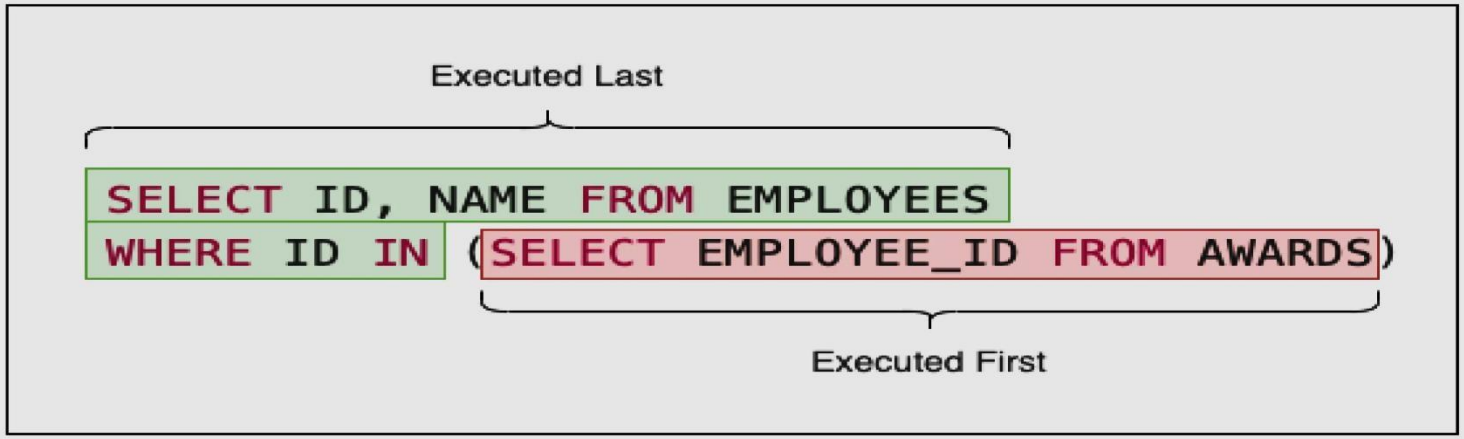
- Nested queries in SQL can be classified into two different types:
  1. Independent Nested Queries
  2. Co-related Nested Queries

1. Independent Nested Queries

- In independent nested queries, the execution order is from the innermost query to the outer query.
- The result of the inner query is used by the outer query. Operators such as **IN**, **NOT IN**, **ALL**, and **ANY** are used to write independent nested queries.
- The **IN** operator checks if a column value in the outer query's result is **present** in the inner query's result. The final result will have rows that satisfy the **IN** condition.
- The **NOT IN** operator checks if a column value in the outer query's result is **not present** in the inner query's result. The final result will have rows that satisfy the **NOT IN** condition.

1. Independent Nested Queries

- The **ALL** operator compares a value of the outer query's result with **all the values** of the inner query's result and returns the row if it matches all the values.
- The **ANY** operator compares a value of the outer query's result with all the inner query's result values and returns the row if there is a match with **any value**.



2. Co-related Nested Queries

- In co-related nested queries, the inner query **uses** the values from the outer query so that the inner query is executed for every row processed by the outer query.
- The co-related nested queries run slowly because the inner query is executed for every row of the outer query's result.

How to Write Nested Query in SQL?

- We can write a nested query in SQL by nesting a **SELECT** statement within another **SELECT** statement.
- The outer **SELECT** statement uses the result of the inner **SELECT** statement for processing.
- The general syntax of nested queries will be:

```
SELECT column_name [, column_name ]
FROM table1 [, table2 ]
WHERE column_name OPERATOR
(
    SELECT column_name [, column_name ]
    FROM table1 [, table2 ]
    [WHERE]
)
```

Examples of Nested Query in SQL

```
CREATE TABLE employee (
    id NUMBER PRIMARY KEY,
    name VARCHAR2(100) NOT NULL,
    salary NUMBER NOT NULL,
    role VARCHAR2(100) NOT NULL
);
```

```
CREATE TABLE awards(
    id NUMBER PRIMARY KEY,
    employee_id NUMBER NOT NULL,
    award_date DATE NOT NULL
);
```

Examples of Nested Query in SQL

EMPLOYEE

- INSERT INTO employees VALUES (1, 'Augustine Hammond', 10000, 'Developer');
- INSERT INTO employees VALUES (2, 'Perice Mundford', 10000, 'Manager');
- INSERT INTO employees VALUES (3, 'Cassy Delafoy', 30000, 'Developer');
- INSERT INTO employees VALUES (4, 'Garwood Saffen', 40000, 'Manager');
- INSERT INTO employees VALUES (5, 'Faydra Beaves', 50000, 'Developer');

AWARDS:

- INSERT INTO awards VALUES(1, 1, TO_DATE('2022-04-01', 'YYYY-MM-DD'));
- INSERT INTO awards VALUES(2, 3, TO_DATE('2022-05-01', 'YYYY-MM-DD'));

Examples of Nested Query in SQL

Employees

id	name	salary	role
1	Augustine Hammond	10000	Developer
2	Perice Mundford	10000	Manager
3	Cassy Delafoy	30000	Developer
4	Garwood Saffen	40000	Manager
5	Faydra Beaves	50000	Developer

Examples of Nested Query in SQL

Awards

id	employee_id	award_date
1	1	2022-04-01
2	3	2022-05-01

1. Independent Nested Queries

Example 1: IN

- Select all employees who won an award.

```
SELECT id, name FROM employees
WHERE id IN (SELECT employee_id FROM awards);
```

Output

id	name
1	Augustine Hammond
3	Cassy Delafoy

1. Independent Nested Queries

Example 2: NOT IN

- Select all employees who never won an award.

```
SELECT id, name FROM employees
WHERE id NOT IN (SELECT employee_id) FROM awards);
```

Output

id	name
2	Perice Mundford
4	Garwood Saffen
5	Faydra Beaves

Examples of Nested Query in SQL

Employees

id	name	salary	role
1	Augustine Hammond	10000	Developer
2	Perice Mundford	10000	Manager
3	Cassy Delafoy	30000	Developer
4	Garwood Saffen	40000	Manager
5	Faydra Beaves	50000	Developer

1. Independent Nested Queries

Example 3: ALL

- Select all **Developers** who earn more than all the **Managers**

```
SELECT * FROM employees
WHERE role = 'Developer'
AND salary > ALL (
    SELECT salary FROM employees WHERE role = 'Manager'
);
```

Output

id	name	salary	role
5	Faydra Beaves	50000	Developer

Examples of Nested Query in SQL

Employees

id	name	salary	role
1	Augustine Hammond	10000	Developer
2	Perice Mundford	10000	Manager
3	Cassy Delafoy	30000	Developer
4	Garwood Saffen	40000	Manager
5	Faydra Beaves	50000	Developer

1. Independent Nested Queries

Example 4: ANY

- Select all **Developers** who earn more than any **Manager**

```
SELECT * FROM employees
WHERE role = 'Developer'
AND salary > ANY (
    SELECT salary FROM employees WHERE role = 'Manager'
);
```

Output

id	name	salary	role
5	Faydra Beaves	50000	Developer
3	Cassy Delafoy	30000	Developer

2. Co-related Nested Queries

- Select all employees whose salary is above the average salary of employees in their role.

```
SELECT * FROM employees emp1
WHERE salary > (
    SELECT AVG(salary) FROM employees emp2
    WHERE emp1.role = emp2.role
);
```

Output

id	name	salary	role
4	Garwood Saffen	40000	Manager
5	Faydra Beaves	50000	Developer

Nested and Complex Queries: Summary

- A nested query in SQL contains a query inside another query, and the outer query will use the result of the inner query.
- We can classify nested queries into independent and co-related nested queries.
- In independent nested queries, the order of execution is from the innermost query to the outermost query
- In co-related nested queries, the inner query uses the values from the outer query so that the inner query is executed for every row processed by the outer query
- Co-related nested query runs slow when compared with independent nested query.

Predicates

- A Predicate in DBMS is a condition expression which evaluates and results in Boolean value either true or false which enables decision making in retrieving and manipulating a record.
- A predicate is a condition that is specified for:
  - Filtering the data using the ***WHERE*** clause,
  - Pattern matching in ***LIKE*** operator,
  - Specifying a set of list for using ***IN*** operator,
  - Manipulating a range of values using ***BETWEEN*** operator, etc.

Predicates: Example

Consider a sample table 'emp'

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	–	17-NOV-81	5000	–	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	500	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	500	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	–	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	–	20
7369	SMITH	CLERK	7902	17-DEC-80	800	500	20

Predicates: Example

Filtering the data using the ***WHERE*** clause/ Comparison Predicate

The predicate in where clause

```
select * from emp
where [job='MANAGER'];
```

O/P

```
3 rows selected.
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	–	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	–	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20

Predicates: Example

Consider a sample table 'emp'

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	–	17-NOV-81	5000	–	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	500	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	500	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	–	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	–	20
7369	SMITH	CLERK	7902	17-DEC-80	800	500	20

Predicates: Example

Specifying a set of list for using ***IN***

Example

```
select empno,job,sal,hiredate
from emp
where [ename in('SCOTT','FORD','SMITH','JONES')];
```

O/P

```
4 rows selected
```

EMPNO	JOB	SAL	HIREDATE
7566	MANAGER	2975	02-APR-81
7788	ANALYST	3000	19-APR-87
7902	ANALYST	3000	03-DEC-81
7369	CLERK	800	17-DEC-80

Predicates: Example

Consider a sample table 'emp'

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	–	17-NOV-81	5000	–	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	500	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	500	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	–	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	–	20
7369	SMITH	CLERK	7902	17-DEC-80	800	500	20

Predicates: Example

Predicate in **BETWEEN** CLAUSE

Example

```
select empno,job,sal,hiredate
from emp
where [sal between 800 and 2900];
```

O/P

```
3 rows selected
```

EMPNO	JOB	SAL	HIREDATE
7698	MANAGER	2850	01-MAY-81
7782	MANAGER	2450	09-JUN-81
7369	CLERK	800	17-DEC-80

Predicates: Example

Consider a sample table 'emp'

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	–	17-NOV-81	5000	–	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	500	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	500	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	–	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	–	20
7369	SMITH	CLERK	7902	17-DEC-80	800	500	20

Predicates: Example

The predicate in **LIKE** clause

Example

```
select empno,ename,hiredate,sal,job
from emp
where [ename like 'S%'];
```

O/P

```
2 rows selected
```

EMPNO	ENAME	HIREDATE	SAL	JOB
7788	SCOTT	19-APR-87	3000	ANALYST
7369	SMITH	17-DEC-80	800	CLERK

Predicates: Example

Consider a sample table 'emp'

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	–	17-NOV-81	5000	–	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	500	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	500	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	–	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	–	20
7369	SMITH	CLERK	7902	17-DEC-80	800	500	20

Predicates: Example

Predicate in **IS NULL** clause

```
select * from emp
where [comm is null]
```

O/P

```
4 rows selected
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	–	17-NOV-81	5000	–	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	–	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	–	20

Predicates: Example

Consider a sample table 'emp'

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	–	17-NOV-81	5000	–	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	500	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	500	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	–	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	–	20
7369	SMITH	CLERK	7902	17-DEC-80	800	500	20

Predicates: Example

The predicate NOT clause

Example

```
select * from emp
where [sal NOT between 800 and 2900 ];
```

O/P

```
4 rows selected
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	–	17-NOV-81	5000	–	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	–	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	–	20

Arithmetic Operators

- To perform various arithmetic operations, we can use these operators
- A comparison operator is a mathematical symbol used to compare two values in mathematical expression
- The result of an arithmetic operators can be any numeric value
- The various comparison operators are enlisted as given in

Operator	
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder

Arithmetic Operators

- **Find all Faculty not teaching 'DT’**

SELECT * FROM Faculty

WHERE Subject <> 'DT’;

- **Find all Faculty teaching more than 10 hours.**

SELECT * FROM Faculty WHERE Hours >10;

Comparison Operators

- To compare attribute value of tuple with arithmetic comparison
- A comparison operator is a mathematical symbol used to compare two values in mathematical expression
- Comparison operators in conditions will be used to evaluate expression. The result of a comparison can be TRUE, FALSE, or UNKNOWN.

Operators	
=	Equal To
<	Less Than
<=	Less Than Equal To
>	Greater Than
>=	Greater Than Equal To
<>	Not equal to

Comparison Operators

Examples

- **Find all Faculty not teaching 'DT’**

SELECT * FROM Faculty WHERE Subject <> ‘DT’;

- **Find all Faculty teaching more than 10 hours.**

SELECT * FROM Faculty WHERE Hours>10 ;

Logical Operators

- The Logical operators will accept expression and return result as true or false to combine one or more true or false values.

Operators	
AND	Logical AND compares between two Booleans expressions to returns true when both expressions are true otherwise false.
OR	Logical OR compares between two Booleans expressions to return true when one of the expression is true otherwise false.
NOT	Not takes a single Boolean expression and changes its value from false to true or from true to false

Logical Operators

Examples

- **Find all faculty not teaching 'DT' and taught hours more than 10**

SELECT * FROM Faculty

WHERE Subject <> 'DT' **AND** Hours > 10

- **Find all Faculty teaching more than 10 hours or taught hours more than 10**

SELECT * FROM Faculty

WHERE Subject <> 'DT' **OR** Hours > 10

Triggers

- A trigger is a **procedure** that is automatically invoked by the DBMS in response to specific alteration database or a table in database.
- Triggers are stored in database as a simple database object.
- A database that has a set of associated triggers is called an **active database.**
- A database trigger enables DBA (Database Administrators) to create additional relationships between separate databases.

Triggers

- **Need/Purpose of Triggers**
  - To generate data automatically
  - Validate input data
  - Replicate data to different files to achieve data consistency

Components of Trigger (E-C-A Model)

- **Event (E):** SQL statement that causes the trigger to fire (or activate). This event may be insert, update or delete operation database table.
- **Condition (C)** : A condition that must be satisfied for execution of trigger.
- **Action (A)** : This is code or statement that execute when triggering condition is satisfied and trigger is activated on database table.

Triggers

Trigger syntax

```
CREATE [OR REPLACE] TRIGGER <Trigger Name>
[<ENABLE | DISABLE>]
<BEFORE|AFTER>
<INSERT|UPDATE | DELETE>
ON <Table_ Name>
        [FOR EACH ROW]
DECLARE
        <Variable_ Definitions>
BEGIN
        <TriggerCode> ;
END;
```

Triggers

- **Trigger Parameters**

OR REPLACE	If trigger is already present then drop and recreate the trigger
< Trigger Name>	Name of trigger to be created.
BEFORE	Indicates that trigger is to be fired before the triggering event occurs
AFTER	Indicates that trigger is to be fired After the triggering event occurs
INSERT	Indicates that trigger is to be fired whenever insert statement adds a row to table
UPDATE	Indicates that trigger is to be fired whenever Update statement modifies a row in a table
DELETE	Indicates that trigger is to be fired whenever delete statement removes a row from table

Triggers

- **Trigger Parameters**

FOR EACH ROW	Trigger will be fired only once for each row.
WHEN	Contains condition that must be satisfied to execute trigger
<trigger code>	Code to be executed whenever triggering event occurs

Trigger Types

1. **Row level Triggers**

- A row level trigger is fired each time the table is affected by the triggering statement
- For example, if an UPDATE statement changes multiple rows in a table, a row trigger is fired once for each row affected by the UPDATE statement
- If a triggering statement do not affect any row then a row trigger will not run only.
- IF **FOR EACH ROW** clause is written that means trigger is row level trigger.

Trigger Types

2. **Statement level triggers**

- A statement level trigger is fired only once on behalf of the triggering statement, irrespective of the number of rows in the table that are affected by the triggering statement.
- This trigger executes once even if no rows are affected.
- For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger fired only one time

Trigger Classification

- The classification of trigger is based on various parameters:

1. **Classification based on the timing**
   a) BEFORE Trigger: This trigger is fired before the occurrence of specified event.
   b) AFTER Trigger: This trigger is fired after the occurrence of specified event.
2. **Classification based on the level**
   b) STATEMENT level Trigger : This trigger is fired for once for the specified event statement.
   c) ROW level Trigger : This trigger is fired for all the records which are affected in the specified event. (only for DML)

Trigger Classification

3. **Classification based on the Event**
   a) DML Trigger : This trigger fires when the DML event such as INSERT/UPDATE / DELETE is specified
   b) DDL Trigger: This trigger fires when the DDL event such as CREATE or ALTER is specified
   c) DATABASE Trigger: This trigger fires when the database event such as LOGON/ LOGOFF , STARTUP/SHUTDOWN) is specified.

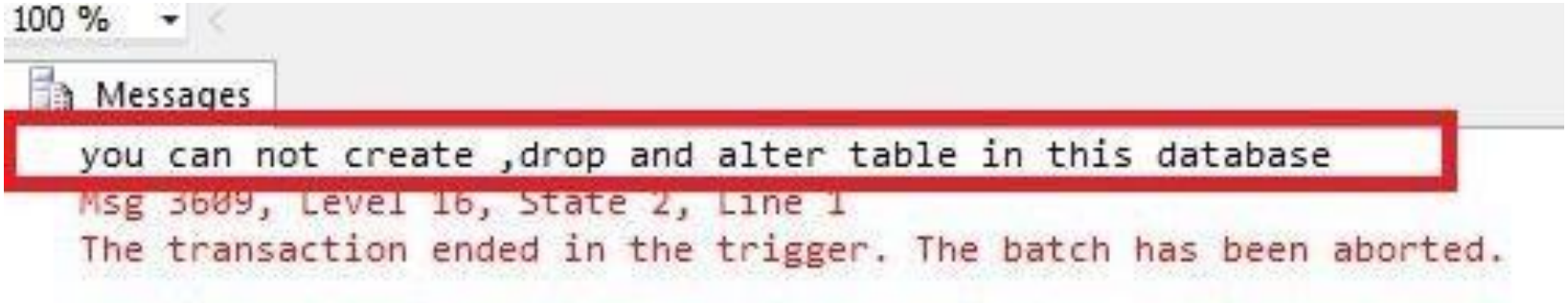
DDL Trigger

- The DDL triggers are fired in response to DDL (Data Definition Language) command events that start with Create, Alter, and Drop, such as Create_table, Create_view, drop_table, Drop_view, and Alter_table.

```
create trigger saftey
on database
for
create_table,alter_table,drop_table
as
print'you can not create ,drop and alter table in this database'
rollback;
```

DDL Trigger

- When we create, alter, or drop any table in a database, then the following message appears,



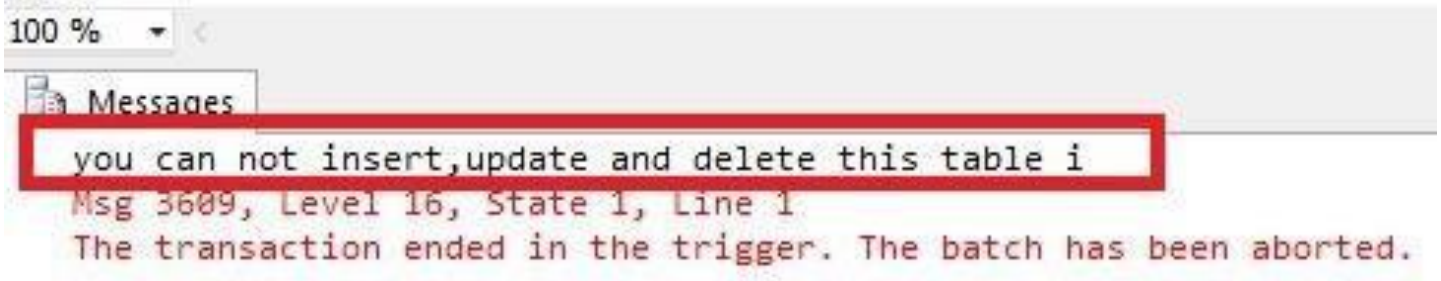
DML Trigger

- The DML triggers are fired in response to DML (Data Manipulation Language) command events that start with Insert, Update, and Delete.
- Like insert_table, Update_view and Delete_table.

```
create trigger deep
on emp
for
insert,update,delete
as
print'you can not insert,update and delete this table i'
rollback;
```

DML Trigger

- When we insert, update, or delete a table in a database, then the following message appears,



Trigger Example: Row Level Trigger

- Creating a trigger on employee table whenever a new employee added, a comment is written in EmpLog table

```
mysql> CREATE OR REPLACE TRIGGER AutoRecruit
AFTER INSERT ON EMP
FOR EACH ROW
BEGIN
        INSERT into EmpLog values(“Employee inserted”);
END;
```

Trigger created

Trigger Example

mysql>INSERT into EMP (1, ‘abc’, ‘manager’, 30000);

1 row created

mysql> SELECT * from EmpLog;

STATUS

Employee inserted

Trigger Example:2

```
CREATE TABLE Student(
studentID INT NOT NULL AUTO_INCREMENT,
FName VARCHAR(20),
LName VARCHAR(20),
Address VARCHAR(30),
City VARCHAR(15),
Marks INT,
PRIMARY KEY(studentID)
);
```

DESC Student;

StudentID	Fname	Lname	Address	City	Marks
INT	Varchar(20)	Varchar(20)	Varchar(30)	Varchar(15)	INT

Trigger Example:2

- ***Before Insert***

```
CREATE TRIGGER calculate
before INSERT
ON student
FOR EACH ROW
SET new.marks = new.marks+100;
```

- Here when we insert data into the student table automatically the trigger will be invoked.
- The trigger will add 100 to the marks column into the student column.

Trigger Example:2

After Insert

- To use this variant we need one more table i.e, Percentage where the trigger will store the results. Use the below code to create the Percentage Table.

```
create table Final_mark(
per int );
```

```
CREATE TRIGGER total_mark
after insert
ON student
FOR EACH ROW
insert into Final_mark values(new.marks);
```

Here when we insert data to the table, *total_mark trigger* will store the result in the Final_mark table.

Trigger Operations

- Data dictionary for triggers
- Dropping triggers
- Disabling Triggers

Trigger Operations

Data dictionary for triggers

- Once triggers are created their definitions can be viewed by selecting it from system tables as shown as follows:

SQL>Select * From User_Triggers

Where Trigger Name= '<Trigger_Name>' ;

- This statement will give you all properties of trigger including trigger code as well.

Dropping Triggers

- To remove trigger from database we use command DROP

SQL> Drop trigger <Trigger_Name>;

Trigger Operations

Disabling Triggers

- To deactivate trigger temporarily this can be activated again by enabling it.

MysQL> Alter trigger <Trigger_ Name> {disable | enable};

Trigger Advantages

- Triggers are useful for enforcing referential integrity, which preserves the defined relationships between tables when you add, update or delete the rows in those tables.
- Make sure that a column is filled with default information.
- After finding that the new information is inconsistent with the database, raise an error that will cause the entire transaction to roll back.

Trigger Disadvantages

- **Invisible from client applications –** Basically MySQL triggers are invoked and executed invisible from the client applications hence it is very much difficult to figure out what happens in the database layer.
- **Impose load on server –** Triggers can impose a high load on the database server.
- **Not recommended for high velocity of data –** Triggers are not beneficial for use with high-velocity data i.e. the data when a number of events per second are high. It is because in case of high-velocity data the triggers get triggered all the time.

Exercise: 1 Query

- **Employee**(Empid,Fname,Lname,Email,Phoneno,Hiredate,Jobid, Salary,Mid,Did)
- **Departments** (Did,Dname,Managerid,Locationid)
- **Locations** (Locationid,Streetadd,Postalcode,City)

- Write SQL Queries for following:

1. List employees having a manager who works for department based in U.S.
2. Display details of all employees in Finance department
3. Give 10% hike to all employees in Did 20
4. Display employee details whose salary is within range 1000 and 3000
5. Display all employee information whose first name starts with ‘R’ in descending order of their salary

Solution

1. **Mysql>**

```
SELECT * FROM Employee e
INNER JOIN Departments d ON e.did=d.did
INNER JOIN Locations l on l.locationid=d.locationid
WHERE city = 'U.S.';
```

2. **Mysql>**

```
SELECT * FROM Employee e
INNER JOIN Departments d ON e.did=d.did
WHERE Dname='Finance'
```

Solutio

3. **Mysql>**

UPDATE Employees
SET salary = 1.1* salary
WHERE did=20;

4. **Mysql>**

SELECT * FROM Employees
WHERE Salary BETWEEN 1000 AND 3000;

5. **Mysql>**

SELECT * FROM Employees
WHERE Fname LIKE ‘R%’
ORDERBY Salary DESC;

Exercise:2 Query

- **Employee**(eid,ename,address,city)
- **Works**(eid,cid,salary)
- **Company**(cid,cname,city)

- Write SQL Queries for:

1. Modify database so that John now lives in Mumbai
2. Find employees who live in same city as company for which they work
3. Give all employees of “AZ corporation” where there is increase in salary by 15%
4. Delete all tuples in works relation for employees of small bank corporation

Solution

1. **Mysql>**

   Update

   Employee SET

   city='Mumbai'

   WHERE ename='John';

2. **Mysql>**

   Select ename FROM Employee e, Company c, works w

   WHERE e.eid=w.eid

   AND w.cid=c.cid

   AND e.city=c.city;

Solution

3. **Mysql>**

```
UPDATE Works
SET Salary = 1.15*salary
WHERE eid IN(SELECT eid FROM Employee e, Works w, Company c
WHERE e.eid=w.eid AND w.cid=c.cid AND cname='AZ Corporation'));
```

4. **Mysql>**

```
DELETE FROM Employee
WHERE eid IN(SELECT eid FROM Works
WHERE cid IN(SELECT cid FROM Company WHERE cname='Small
bank corporation'));
```

Exercise:3

Query

- **For given database, write SQL queries:**

PERSON(driver_id#, name, address)

CAR(license, model, year)

ACCIDENT(report_no, date, location)

OWNS(driver_id#, license)

PARTICIPATED (driver_id, car, report_number, damage_amount)

1. Add new accident to database
2. Delete ‘Santro’ belonging to ‘John Smith’

Solution

1. Mysql>

```
INSERT INTO Accident(report_no, adate, location)
VALUES ('111','01/02/2023','PUNE');
```

2. Mysql>

```
DELETE FROM CAR
WHERE Model ='Santro'
AND
License IN(SELECT license FROM Owns
WHERE Driver_id IN
(SELECT driver_id FROM person
WHERE name= 'John Smith'));
```

Thank You!!