Unit - 2

SQL, Intermediate SQL and Relational Database Design:
SQL: Overview of SQL, DDL, integrity Constraints, DML, Set Operations, Null values, aggregate Functions, Sub-queries
Intermediate SQL: Joins, Views, Indexes, Abstract Data type

---

* SQL - Structured Query language

- Developed by IBM in the 1970's, Originally called Sequel
- The Standard relational-database language
- SQL is used to interact with a database to manage and retrieve data
- SQL is a non-procedural language

Data types

• Integer • Float • Char • Varchar2 • Number • date

1) Integer : Used to store whole numbers (without decimal Points)
Ex: 10, -50, 1000

2) Float : Used to store decimal numbers (Floating-Point numbers)
Ex: 3.14, -0.75, 100.25

3) Char : • Stored fixed-length character Strings
• If the defined length is 5, it always stores 5 characters (even if fewer characters are used, spaces are padded)
Ex: CHAR(5) stores 'ABC__'

4) Varchar2 : Variable-length character data type
• Unlike CHAR, it does not pad spaces if the value is shorter than the defined length
Ex: VARCHAR2(10) can store 'Hello' (takes only 5 bytes, not 10).

5] NUMBER : Used in Oracle database to store both integers and floating point numbers

Ex: Number(5) → Can store up to 5-digit integers (99999)
Number(7,2) → Can store up to 7 digit with 2 decimal places (12345.67)

6] DATE : Stores date and time values.
Default format in SQL : 'YYYY-MM-DD'
Ex: '2025-03-01' (March 1, 2025)

* DML is Data Manipulation Language Statements

Example: Select - Retrieves data from a table
Insert - insert data into a table
Update - Modifies existing records
Delete - Removes records from table

* DDL is Data Definition Language
• Is used to define, modify and delete database structures

Create - Creates a new database object
Alter - Modifies an existing database object
DROP - Deletes an existing database object
Truncate - Removes all records from a table but keeps its structure
Rename - Renames a table or column

* DCL (Data Control Language)

• Is used to control access and permissions in a database.
• It is mainly used for security purposes, allowing to grant or revoke privileges to users

Grant - Gives specific privileges to a user
Revoke - Removes specific privileges from a user
Commit - Save work done
Rollback - restore database to original since the last Commit

SQL - Create Table

Syntax: CREATE TABLE table-name (
    Column1 datatype Constraints,
    Column2 datatype Constraints,
    ....
);

Ex: Employe Table

```
CREATE TABLE employees (
    id INT Primary key,
    name Varchar(20),
    Salary Decimal(10,2),
);
```

ALTER TABLE

1. Add a New Column

Syntax: ALTER TABLE table-name
ADD Column-name datatype Constraints;

Ex: ALTER TABLE employees
ADD department VARCHAR(50);

2. Modify an Existing Column

Synatax: ALTER TABLE table-name;
MODIFY Column-name new-datatype);

Ex: ALTER TABLE employees;
MODIFY Salary DECIMAL(12,2);

3. Rename a Column

syntax: ALTER TABLE table-name
RENAME COLUMN old-Column-name To new-column-name

Ex: ALTER TABLE employees
RENAME Column department To dept-name);

4. Drop (Delete) a Column

Syntax: ALTER TABLE table_name
DROP COLUMN Column_name;

Ex: ALTER TABLE employees
DROP Column dept_names;

Note: All data in the column will be lost Permanently

* Difference Between TRUNCATE and DELETE

Both Truncate and Delete are used to remove records from a table, but they work differently

1. Delete Statement
- used to remove specific records (or all records) from a table.
- Rollback is possible (Delete is a DML Command)
- can include a WHERE condition to delete selected rows

2. Truncate Statement
- Removes all rows from a table without a WHERE condition
- Faster than DELETE (Truncate is a DDL Command)
- Cannot be rolled back (Permanent action)

* GROUP BY
- The Group By clause is used in a SELECT Statement to collect data across multiple records and grop the results by one or more Columns
- Sometimes it is required to get information not about each row, but about each group.

* Integrity Constraints in SQL

- Integrity Constraints are rules enforced on table columns to ensure the accuracy and consistency of data in a relational database.

Types of Integrity Constraints

1. Primary Key Constraints

- Ensures that each row in a table is uniquely identified
- A table can have only one primary key
- The column(s) must be unique and NOT NULL

2. Foreign Key Constraints

- Ensures referential integrity between two tables.
- The foreign key column must reference a primary key from another table.

3. Unique Constraint

- Ensures that all values in a column are different
- Unlike Primary key, a table can have multiple UNIQUE colu

4. NOT NULL Constraint

- Prevents NULL values from being entered in a column.

5. Check Constraint

- Ensures that a column meets a specific condition
- Can be used to restrict values entered into a column.

6. Default Constraint

- Assigns a default value to a column if no value is provided

* Set Operations in SQL *

- Set operations in SQL are used to combine results from multiple SELECT queries.

1. UNION 2. UNION ALL 3. INTERSECT

1. UNION

- Combines results from two queries and remove duplicates
- Both queries must have the same number of columns and compatible data types.

Syntax: SELECT Col_1, Col_2 From table1
UNION
Select Col_1, Col_2 from table2

Example: Select name from Students
UNION
Select name from teachers;

Returns a list of unique names from both tables.

2. UNION ALL

- Similar to UNION, but keeps duplicate values
- Faster than UNION since it does not remove duplicates

Syntax: Select Column_1 from table1
UNION ALL
Select Column_1 from table 2;

Example: Select name from students
UNION ALL
Select name from teachers;

Returns all names, including duplicates.

3. Intersect

- Returns only common records present in both queries
- The columns must have the same number and data type

Syntax: Select Col_1 from table1
INTERSECT
Select Col_1 from table2;

Example: Select name from students
INTERSECT
Select name from teachers;

Returns names present in both tables

* NULL Values in SQL

- NULL represents a missing, unknown, or undefined value in a database table.
- It is not the same as 0, an empty string ("") or a space
- NULL means missing or unknown data, not zero or empty
- Use ISNULL or IS NOT NULL to filter NULL values
- Aggregate functions ignore NULL values

* Aggregate Functions

Aggregate Functions perform Calculations on multiple rows and return a single result. These are commonly used with the GROUP BY Clause.

1. Count() - Counts Rows : Counts the number of rows in a Column (excluding NULL values)

Syntax: Select Count (col_name) from table_name;

Example: 1] Select Count (student_id) from students;
→ Counts the total number of student

2] Select Count(*) From students;
→ Counts all rows, including those with NULL values

2. Sum() - Calculates Total : Returns the sum of values in a numeric column

Syntax: Select SUM (col_name) From table_name;

Example: Select Sum (salary) From employees;

returns the total sum of salaries

3. AVG() - Calculates Average
• returns the average (mean) of numeric values

Syntax: Select AVG(col_name) from table_name;

Example: Select AVG(Salary) From employees;
→ returns the average Salary of employees

4. MAX() - Finds Maximum Value
• returns the highest value in a column

Syntax: Select MAX(col_name) from table_name;

Example: Select MAX(Salary) From employees;
→ returns the highest Salary

5. MIN() - Finds Minimum Value (Same as Max())

6. Group By - Aggregating

The Group By clause is used to group data based on a column and apply aggregate functions to each group

Ex: Find total Salary per department

```
Select department, SUM(Salary)
From employees
Group By department;
```

→ Groups employees by department and calculates the total Salary in each department

7. HAVING - Filtering Aggregate Results

Unlike WHERE, which filters individual rows, HAVING filters grouped data

Ex: Find departments where total salary > 50,000

```
Select department, Sum(salary) As total-salary
FROM employees
Group By department
HAVING Sum(Salary) > 50000;
```

→ Displays only department where the total salary exceeds 50,000

Sub-Queries

- A Subquery (also called a nested query) is a query inside another SQL query.

* Types of Subqueries

1] Single-row subqueries:
→ returns one value (used with =, >, <, >=, <=, <>)

Ex: Find employees earning more than the average salary

```
Select name, salary
From employees
WHERE Salary > (Select AVG(salary) From employees);
```

→ The Subquery calculates the average salary, and the other query fetches employees earning above it.

2] Multiple-Row Subqueries
→ A Subquery that returns multiple values (IN, ANY, ALL)

Ex: Find employees working in the same department as 'Ritu'

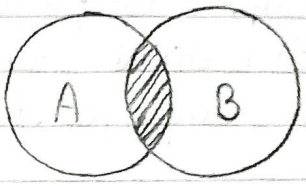
```
Select name, department
from employees
Where department IN (select department from employees
                     WHERE name = 'Ritu');
```

→ The Subquery finds Ritu's department, and the outer query selects employees from the same department

✳ SQL JOINS ✳

- SQL Joins are used to combine rows from two or more tables based on related column.
- Join operations take two tables and return another table as result.

1] INNER Join (Default Join)



→ Returns only the matching records from both tables

employees table

emp_id	name	dept_id
1	Ritu	101
2	Piyu	102
3	Aanu	103
4	Niku	104

departments table

dept_id	dept_name
101	HR
102	IT
103	Finance

Query: Get employee names with their department names

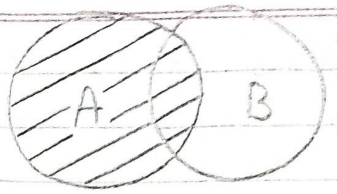
```
Select employees.name, departments.dept_name
FROM employees
INNER JOIN departments
ON employees.dept_id = departments.dept_id;
```

Output:

name	dept_name
Ritu	HR
Piyu	IT
Aanu	Finance

→ Niku is not included because she has no matching department

2] LEFT JOIN



- returns all records from the left table and matching records from the right table.
- If no match is found, NULL is returned

Query: Get all employees and their departments (if any)

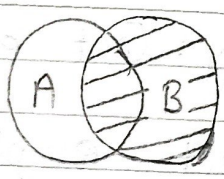
```
Select  employees.name , departments.dept-name
from    employees
LEFT JOIN  departments
ON employees.dept-id = department.dept-id;
```

Output:

name	dept-name
Ritu	HR
Piyu	IT
Aaru	Finance
Niku	NULL

→ Niku appears with NULL because he has no matching department

3] RIGHT JOIN



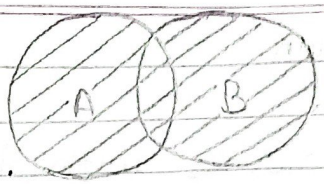
- returns all records from the right table and matching records from the left table
- If no match is found, NULL is returned

query: Get all department and their employees

```
Select  employees.name, department.dept-name
From    employees
RIGHT JOIN  departments
On  employees.dept-id = department.dept-id;
```

name	dept_name
Ritu	HR
Piyu	IT
Aaru	Finance
NULL	Marketing

4] FULL JOIN



- Returns all records from both tables.
  If there's no match, NULL is returned.

Query : Get all employees and all departments

```
Select employees.name , departments.dept_name
From employees
FULL JOIN departments
ON employees.dept_id = departments.dept_id;
```

Output :

name	dept_name
Ritu	HR
Piyu	IT
Aaru	Finance
niku	NULL
NULL	Marketing

→ Both unmatched employees and unmatched departments appear with NULL

5) CROSS JOIN

- Returns the Cartesian product of both tables
- every row from the first table pairs with every row from the second table

Query: Get all employee-department combinations

```
Select employees.name, departments.dept_name
From employees
CROSS JOIN departments;
```

Output :

name	dept_name
Ritu	HR
Ritu	IT
Ritu	Finance

Same for all other name

6] Self Join

- A table joins with itself, treating it as two separate tables

- Ex: Find employees and their managers (assuming manager exists in employees)

```
Select e1.name AS Employee, e2.name AS Manager
From employees e1
Left Join employees e2
ON e1.manager_id = e2.emp_id;
```

Output:

Employee	Manager
Ritu	Piyu
Piyu	Aaru
Aaru	NULL

→ Aaru has no manager (NULL).