**Structured Query Language (SQL)**

**1. Introduction to SQL**

Structured Query Language (SQL) is a standard language for managing and manipulating relational databases. It allows users to create, update, retrieve, and manage data efficiently.

**Key Categories of SQL Commands:**

1. **Data Retrieval Language (DRL):**
   * Focuses on retrieving data from the database.
   * Primary command: SELECT.
   * SELECT \* FROM employees;
2. **Data Definition Language (DDL):**
   * Defines and manages database structures.
   * Commands:

CREATE: Create tables, databases, views.

CREATE TABLE employees (

id INT PRIMARY KEY,

name VARCHAR(50),

position VARCHAR(50),

salary DECIMAL(10, 2)

);

ALTER: Modify structure of database objects.

ALTER TABLE employees ADD COLUMN department VARCHAR(50);

DROP: Remove database objects.

DROP TABLE employees;

1. **Data Manipulation Language (DML):**
   * Manages data within tables.
   * Commands:
     + INSERT: Add new data.
     + INSERT INTO employees (id, name, position, salary)
     + VALUES (1, 'John Doe', 'Manager', 75000);
     + UPDATE: Modify existing data.
     + UPDATE employees SET salary = 80000 WHERE id = 1;
     + DELETE: Remove data.
     + DELETE FROM employees WHERE id = 1;
2. **Data Control Language (DCL):**
   * Controls access to the database.
   * Commands:
     + GRANT: Provide user access.
     + GRANT SELECT, INSERT ON employees TO user\_name;
     + REVOKE: Remove user access.
     + REVOKE INSERT ON employees FROM user\_name;
3. **Transaction Control Language (TCL):**
   * Manages database transactions.
   * Commands:
     + COMMIT: Save changes permanently.
     + COMMIT;
     + ROLLBACK: Undo changes.
     + ROLLBACK;
     + SAVEPOINT: Set a savepoint within a transaction.
     + SAVEPOINT savepoint\_name;

**2. SELECT Commands**

The SELECT command is used to retrieve data from a database. It is one of the most powerful and versatile SQL statements.

**2.1 Projecting Data**

* Specify which columns to display using the SELECT clause.
* SELECT name, position FROM employees;

**2.2 Selecting Data**

* Use the WHERE clause to filter rows based on conditions.
* SELECT \* FROM employees WHERE salary > 50000;

**2.3 Ordering Data**

* Sort data using the ORDER BY clause.
* SELECT \* FROM employees ORDER BY salary DESC;

**2.4 Joining Data**

Joins are used to combine rows from two or more tables based on a related column.

**Types of Joins:**

1. **Inner Join:** Retrieves rows that have matching values in both tables. Commonly used to fetch related data.
2. SELECT employees.name, departments.name AS department
3. FROM employees
4. INNER JOIN departments ON employees.department\_id = departments.id;

Example Tables:

* + Table: employees

| **id** | **name** | **department\_id** |
| --- | --- | --- |
| 1 | John Doe | 101 |
| 2 | Jane Smith | 102 |
| 3 | Alice Lee | 103 |

* + Table: departments

| **id** | **name** |
| --- | --- |
| 101 | HR |
| 102 | IT |
| 104 | Marketing |

* + Result:

| **name** | **department** |
| --- | --- |
| John Doe | HR |
| Jane Smith | IT |

1. **Left Outer Join:** Retrieves all rows from the left table, and matching rows from the right table. If there is no match, NULL is returned for the right table's columns.
2. SELECT employees.name, departments.name AS department
3. FROM employees
4. LEFT JOIN departments ON employees.department\_id = departments.id;

Result:

| **name** | **department** |
| --- | --- |
| John Doe | HR |
| Jane Smith | IT |
| Alice Lee | NULL |

1. **Right Outer Join:** Retrieves all rows from the right table, and matching rows from the left table. If there is no match, NULL is returned for the left table's columns.
2. SELECT employees.name, departments.name AS department
3. FROM employees
4. RIGHT JOIN departments ON employees.department\_id = departments.id;

Result:

| **name** | **department** |
| --- | --- |
| John Doe | HR |
| Jane Smith | IT |
| NULL | Marketing |

1. **Full Outer Join:** Retrieves all rows from both tables. Rows without a match in one table will show NULL for the corresponding columns from the other table.
2. SELECT employees.name, departments.name AS department
3. FROM employees
4. FULL OUTER JOIN departments ON employees.department\_id = departments.id;

Result:

| **name** | **department** |
| --- | --- |
| John Doe | HR |
| Jane Smith | IT |
| Alice Lee | NULL |
| NULL | Marketing |

1. **Cross Join:** Produces the Cartesian product of both tables, combining every row of the first table with every row of the second table.
2. SELECT employees.name, departments.name AS department
3. FROM employees
4. CROSS JOIN departments;

Example:

* + Cartesian product for 3 employees and 3 departments will yield 9 rows.

| **name** | **department** |
| --- | --- |
| John Doe | HR |
| John Doe | IT |
| John Doe | Marketing |
| Jane Smith | HR |
| Jane Smith | IT |
| Jane Smith | Marketing |
| Alice Lee | HR |
| Alice Lee | IT |
| Alice Lee | Marketing |

1. **Recursive Join:** Allows recursive queries, often used with Common Table Expressions (CTEs) to represent hierarchical data.
2. WITH RECURSIVE employee\_hierarchy AS (
3. SELECT id, name, manager\_id
4. FROM employees
5. WHERE manager\_id IS NULL
6. UNION ALL
7. SELECT e.id, e.name, e.manager\_id
8. FROM employees e
9. INNER JOIN employee\_hierarchy eh ON e.manager\_id = eh.id
10. )
11. SELECT \* FROM employee\_hierarchy;

Example Tables:

* + Table: employees

| **id** | **name** | **manager\_id** |
| --- | --- | --- |
| 1 | John Doe | NULL |
| 2 | Jane Smith | 1 |
| 3 | Bob Brown | 2 |

* + Result:

| **id** | **name** | **manager\_id** |
| --- | --- | --- |
| 1 | John Doe | NULL |
| 2 | Jane Smith | 1 |
| 3 | Bob Brown | 2 |

**2.5 Single Row Functions**

Functions that operate on single values and return a single result:

* **String functions:**
* SELECT UPPER(name), LOWER(position) FROM employees;
* **Numeric functions:**
* SELECT ROUND(salary, 2) FROM employees;
* **Date functions:**
* SELECT NOW(), DATE\_ADD(hire\_date, INTERVAL 1 YEAR) FROM employees;

**2.6 Aggregate Functions**

Functions that operate on multiple rows and return a single summary value:

* Examples: SUM(), AVG(), COUNT(), MAX(), MIN().
* SELECT AVG(salary), COUNT(\*) FROM employees;

**2.7 Group By and Having Clauses**

* **Group By:** Groups rows with the same values in specified columns.
* SELECT department\_id, COUNT(\*) FROM employees
* GROUP BY department\_id;
* **Having Clause:** Filters groups based on aggregate functions.
* SELECT department\_id, AVG(salary) FROM employees
* GROUP BY department\_id
* HAVING AVG(salary) > 60000;

**2.8 Subqueries**

A subquery is a query nested inside another query.

**Types of Subqueries:**

1. **Single-row Subquery:** Returns one row.
2. SELECT \* FROM employees WHERE salary = (SELECT MAX(salary) FROM employees);
3. **Multi-row Subquery:** Returns multiple rows.
4. SELECT \* FROM employees WHERE department\_id IN (SELECT id FROM departments WHERE location = 'New York');
5. **Multi-column Subquery:** Returns multiple columns.
6. SELECT \* FROM employees WHERE (department\_id, position) = (SELECT id, title FROM departments WHERE title = 'Engineering');