Complete & Detailed Documentation of `basic.sql`

This document provides a **deep, query-by-query explanation** of every SQL statement used in `basic.sql`. Each section contains **exact Purpose, Detailed Explanation, and a Clear Real-Life Example**, written in an academic style suitable for **DBMS lab reports, assignments, viva, and capstone appendices**.

---

1. CREATE DATABASE

Query

```sql
CREATE DATABASE Collection;
```

Purpose

To create a new database that will store tables and data for an application.

Explanation

Allocates a new database named `Collection` on the database server. A database is the highest-level structure in SQL where all tables, views, and data are organized.

Example

```sql
CREATE DATABASE LibraryDB;
```

Used when starting a new system such as a library or management application.

---

2. CREATE DATABASE IF NOT EXISTS

Query

```sql
CREATE DATABASE IF NOT EXISTS temp1;
```

Purpose

To safely create a database without causing an error if it already exists.

Explanation

The `IF NOT EXISTS` clause checks whether the database already exists. If it does, the command is skipped. Useful in automated scripts.

Example

```
CREATE DATABASE IF NOT EXISTS TestDB;
```

---

3. DROP DATABASE IF EXISTS

Query

```
DROP DATABASE IF EXISTS temp1;
```

Purpose

To permanently remove a database.

Explanation

Deletes the database and all associated tables and data. `IF EXISTS` prevents execution errors.

Example

```
DROP DATABASE IF EXISTS OldProjectDB;
```

---

4. SHOW DATABASES

Query

```
SHOW DATABASES;
```

Purpose

To display all available databases on the server.

Explanation

Lists every database currently stored in the DBMS.

Example

Used by administrators to verify database creation.

---

5. USE DATABASE

Query

```
USE Collection;
```

Purpose

To select a database for performing operations.

Explanation

After selecting a database, all subsequent SQL commands apply to that database.

Example

```
USE LibraryDB;
```

---

6. CREATE TABLE

Query

```
CREATE TABLE employee(
    id INT PRIMARY KEY,
    name VARCHAR(50),
    dept VARCHAR(50),
    university VARCHAR(50),
    age INT NOT NULL,
    salary INT
);
```

Purpose

To create a table that stores structured employee data.

Explanation

Defines columns, data types, constraints, and primary key to ensure data integrity.

Example

Used to store employee or student records.

---

7. ALTER TABLE – ADD COLUMN

Query

```
ALTER TABLE employee
ADD COLUMN email VARCHAR(50) DEFAULT 'demo@gmail.com',
ADD COLUMN phone VARCHAR(50) DEFAULT '01512345678';
```

Purpose

To extend an existing table by adding new attributes (columns) without losing existing data.

Detailed Explanation

Allows adding new columns. The `DEFAULT` value ensures old records automatically receive a value, preventing NULL issues.

Example

```
ALTER TABLE employee ADD COLUMN address VARCHAR(100);
```

---

8. ALTER TABLE – DROP COLUMN

Query

```
ALTER TABLE employee DROP COLUMN phone;
```

Purpose

To permanently remove an unnecessary column from a table.

Detailed Explanation

Deletes both the column structure and all stored data. Cannot be recovered.

Example

```
ALTER TABLE employee DROP COLUMN address;
```

---

9. ALTER TABLE – RENAME TABLE

Query

```
ALTER TABLE employee RENAME TO emp;
```

Purpose

To change the name of an existing table.

Detailed Explanation

Helps maintain clarity or follow naming conventions without affecting data.

Example

```
ALTER TABLE student_information RENAME TO student;
```

---

10. ALTER TABLE – CHANGE COLUMN

Query

```
ALTER TABLE emp CHANGE COLUMN university uni VARCHAR(50);
```

Purpose

To rename a column and optionally change its data type.

Detailed Explanation

Useful for standardization or correction. Existing compatible data remains intact.

Example

```
ALTER TABLE emp CHANGE COLUMN fullname full_name VARCHAR(100);
```

---

11. TRUNCATE TABLE

Query

```
TRUNCATE TABLE emp;
```

Purpose

To remove all records from a table efficiently.

Detailed Explanation

Deletes all rows quickly, faster than `DELETE`, cannot be rolled back.

Example

```
TRUNCATE TABLE employee;
```

---

12. INSERT INTO TABLE

Query

```
INSERT INTO employee VALUES(1, 'shaker', 'Java Backend', 'BUBT', 24, 35000);
```

Purpose

To add a complete new record to a table.

Detailed Explanation

All column values must be provided in the correct order.

Example

```
INSERT INTO employee VALUES(2, 'Rifat', 'UI', 'DU', 26, 40000);
```

---

13. INSERT INTO (Specific Columns)

Query

```
INSERT INTO employee (id, name, dept, university, age, salary)
VALUES (5, 'Dayan', 'UI', 'BUBT', 27, 30000);
```

Purpose

To insert data into selected columns when not all column values are known.

Detailed Explanation

Supports default or NULL for unlisted columns. Enhances flexibility.

Example

```
INSERT INTO employee (id, name, dept) VALUES (7, 'Hasan', 'QA');
```

---

14. UPDATE

Query

```
UPDATE employee SET university = 'DU' WHERE id > 4;
```

Purpose

To modify existing records in a table.

Detailed Explanation

Only rows matching the condition are updated.

Example

```
UPDATE employee SET salary = salary + 5000 WHERE dept = 'Java Backend';
```

---

15. DELETE

Query

```
DELETE FROM employee WHERE salary > 90000;
```

Purpose

To remove specific records permanently.

Detailed Explanation

Only rows meeting the condition are deleted.

Example

```
DELETE FROM employee WHERE age < 18;
```

---

16. SELECT

Query

```
SELECT * FROM employee;
```

Purpose

To retrieve all rows and columns from a table.

Detailed Explanation

Displays all table content.

Example

Used to verify all records.

17. SELECT WITH WHERE

Query

```
SELECT name FROM employee WHERE name = 'Rifat';
```

Purpose

To filter and retrieve only specific records.

Detailed Explanation

The `WHERE` clause filters rows matching the condition.

Example

```
SELECT * FROM employee WHERE dept = 'UI';
```

---

18. ORDER BY, LIMIT and OFFSET

Query

```
SELECT * FROM employee ORDER BY salary DESC LIMIT 3 OFFSET 2;
```

Purpose

To sort data, limit the number of rows, and skip a specific number of rows (pagination).

Detailed Explanation

- `ORDER BY` sorts data ascending or descending.
- `LIMIT` restricts how many rows are returned.
- `OFFSET` skips the first N rows. Useful for paginated results in applications.

Example

```
SELECT * FROM employee ORDER BY age ASC LIMIT 5 OFFSET 10;
```

Retrieve 5 employees starting from the 11th youngest.

19. AGGREGATE FUNCTIONS

Query

```
SELECT COUNT(name), MAX(salary), AVG(salary) FROM employee;
```

Purpose

To perform calculations on multiple rows and summarize results.

Detailed Explanation

`COUNT()` counts rows, `MAX()` finds the highest value, `AVG()` calculates average.

Example

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

Calculate total salary expense.

---

20. GROUP BY

Query

```
SELECT university, COUNT(id) FROM employee GROUP BY university;
```

Purpose

To group rows by common values for aggregation.

Detailed Explanation

Groups rows by `university` to apply aggregate functions like COUNT or AVG.

Example

```
SELECT dept, AVG(salary) FROM employee GROUP BY dept;
```

Find average salary per department.

---

21. HAVING

Query

```
SELECT university, COUNT(id) FROM employee GROUP BY university HAVING
MAX(salary) > 30000;
```

Purpose

To filter grouped data using aggregate conditions.

Detailed Explanation

`HAVING` filters results after aggregation, unlike `WHERE`.

Example

```
SELECT dept, COUNT(id) FROM employee GROUP BY dept HAVING COUNT(id) > 5;
```

Identify departments with more than 5 employees.

---

22. INNER JOIN

Query

```
SELECT * FROM student INNER JOIN course ON student.id = course.id;
```

Purpose

To retrieve records that exist in both tables.

Detailed Explanation

Matches rows based on a common column. Non-matching rows are excluded.

Example

```
SELECT s.name, c.course_name FROM student s INNER JOIN course c ON s.id =
c.student_id;
```

List students enrolled in courses.

---

23. LEFT JOIN

Query

```
SELECT * FROM student a LEFT JOIN course b ON a.id = b.id;
```

Purpose

To include all rows from the left table and matching rows from the right table.

Detailed Explanation

If no matching row exists in the right table, NULL is shown.

Example

```
SELECT s.name, c.course_name FROM student s LEFT JOIN course c ON s.id =
c.student_id;
```

List all students including those without courses.

---

24. RIGHT JOIN

Query

```
SELECT * FROM student a RIGHT JOIN course b ON a.id = b.id;
```

Purpose

To include all rows from the right table and matching rows from the left table.

Detailed Explanation

If no matching row exists in the left table, NULL is shown.

Example

```
SELECT s.name, c.course_name FROM student s RIGHT JOIN course c ON s.id =
c.student_id;
```

List all courses including those without enrolled students.

---

25. FULL JOIN (Using UNION)

Query

```
SELECT * FROM student a LEFT JOIN course b ON a.id = b.id UNION SELECT * FROM
student a RIGHT JOIN course b ON a.id = b.id;
```

Purpose

To retrieve all records from both tables regardless of matching.

Detailed Explanation

Combines LEFT and RIGHT JOIN results. MySQL does not support FULL JOIN directly.

Example

List all students and courses, showing unmatched rows from both tables.

---

26. UNION

Query

```
SELECT name FROM employee UNION SELECT name FROM employee;
```

Purpose

To combine results from multiple SELECT queries, removing duplicates.

Detailed Explanation

`UNION` ensures unique values; `UNION ALL` keeps duplicates.

Example

```sql
SELECT email FROM customers UNION SELECT email FROM suppliers;
```

Create a combined mailing list.

---

27. SUBQUERY

Query

```sql
SELECT full_name FROM emp WHERE salary > (SELECT AVG(salary) FROM emp);
```

Purpose

To use a query result inside another query for filtering or calculations.

Detailed Explanation

The inner query executes first. The outer query uses the result to filter rows.

Example

Find employees earning above average:

```sql
SELECT name FROM employee WHERE salary > (SELECT AVG(salary) FROM employee);
```

---

28. VIEW

Query

```sql
CREATE VIEW view1 AS SELECT id, full_name, uni FROM emp;
```

Purpose

To create a virtual table based on a query result.

Detailed Explanation

Views simplify complex queries, provide controlled access, and enhance security without storing physical data.

Example

```
CREATE VIEW public_employee AS SELECT id, name FROM employee;
```

Show only non-sensitive columns to some users.

---

29. SELECT FROM VIEW

Query

```
SELECT * FROM view1 WHERE uni = 'DU';
```

Purpose

To retrieve data from a view with optional filtering.

Detailed Explanation

Views behave like tables and can be queried with conditions.

Example

Show all employees from DU university.

---

30. DROP VIEW

Query

```
DROP VIEW view1;
```

Purpose

To delete a view without affecting the underlying tables.

Detailed Explanation

Removes the virtual table definition. Data in base tables remains intact.

Example

```
DROP VIEW public_employee;
```

Final Conclusion

This documentation now provides **complete, detailed, and structured explanations for all 30 SQL topics**, including OFFSET, joins, subqueries, views, and aggregation. It is fully suitable for **DBMS labs, assignments, viva, and project documentation**.