SQL Interview Preparation Guide

give SQL notes for freshers interview in detail in more deatils with code

Here's a comprehensive SQL guide for freshers with essential concepts, explanations, and example queries. This will cover fundamental SQL operations, queries, and important points often discussed in interviews.

---

1. Introduction to SQL

- **SQL (Structured Query Language)** is used to communicate with relational databases.
- SQL commands are divided into several categories:
  - **DDL (Data Definition Language)**: `CREATE`, `ALTER`, `DROP`
  - **DML (Data Manipulation Language)**: `SELECT`, `INSERT`, `UPDATE`, `DELETE`
  - **DCL (Data Control Language)**: `GRANT`, `REVOKE`
  - **TCL (Transaction Control Language)**: `COMMIT`, `ROLLBACK`

---

2. Basic SQL Commands

a. SELECT

- The `SELECT` statement is used to fetch data from a database table.

```sql
SELECT column1, column2, ...
FROM table_name;
```

- **Example:**

```sql
```

```sql
SELECT name, age FROM Employees;
```

b. WHERE

- The `WHERE` clause filters records based on conditions.

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

- **Example**:

```sql
SELECT * FROM Employees WHERE age > 30;
```

c. INSERT

- The `INSERT INTO` statement adds new rows to a table.

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

- **Example**:

```sql
INSERT INTO Employees (name, age, department)
VALUES ('John Doe', 29, 'HR');
```

d. UPDATE

- The `UPDATE` statement modifies existing records in a table.

```sql

```

```sql
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

- **Example**:

  ```sql
  UPDATE Employees
  SET department = 'Finance'
  WHERE name = 'John Doe';
  ```

e. DELETE

- The `DELETE` statement removes rows from a table.

```sql
DELETE FROM table_name WHERE condition;
```

- **Example**:

  ```sql
  DELETE FROM Employees WHERE age < 25;
  ```

---

3. Common SQL Functions

a. Aggregate Functions

- **COUNT**: Counts the number of rows.

  ```sql
  SELECT COUNT(*) FROM Employees;
  ```

- **SUM**: Adds up all values in a column.

  ```sql
  ```

```sql
SELECT SUM(salary) FROM Employees;
```

- **AVG**: Calculates the average.

```sql
SELECT AVG(salary) FROM Employees;
```

- **MIN/MAX**: Finds the smallest or largest value.

```sql
SELECT MIN(age), MAX(age) FROM Employees;
```

b. GROUP BY

- The `GROUP BY` statement groups rows that have the same values in specified columns.

```sql
SELECT department, COUNT(*)
FROM Employees
GROUP BY department;
```

c. HAVING

- The `HAVING` clause is used to filter groups after aggregation.

```sql
SELECT department, AVG(salary)
FROM Employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

---

4. Joins in SQL

a. Inner Join

- Returns records that have matching values in both tables.

```sql
SELECT Employees.name, Departments.department
FROM Employees
INNER JOIN Departments ON Employees.department_id = Departments.id;
```

b. Left Join

- Returns all records from the left table and the matched records from the right table.

```sql
SELECT Employees.name, Departments.department
FROM Employees
LEFT JOIN Departments ON Employees.department_id = Departments.id;
```

c. Right Join

- Returns all records from the right table and the matched records from the left table.

```sql
SELECT Employees.name, Departments.department
FROM Employees
RIGHT JOIN Departments ON Employees.department_id = Departments.id;
```

d. Full Outer Join

- Returns all records when there is a match in either left or right table.

```sql
SELECT Employees.name, Departments.department
FROM Employees
FULL OUTER JOIN Departments ON Employees.department_id = Departments.id;
```

---

5. Subqueries

- A **subquery** is a query within another query.
  - **Example**:

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

---

6. Indexes

- **Indexes** are used to speed up the retrieval of data.
  - **Creating an Index**:

```sql
CREATE INDEX idx_emp_name ON Employees(name);
```

- **Dropping an Index**:

```sql
DROP INDEX idx_emp_name;
```

---

7. Constraints

Constraints enforce data integrity in SQL tables:

a. Primary Key

- Uniquely identifies each record in a table.

```sql
```

```sql
CREATE TABLE Employees (
  id INT PRIMARY KEY,
  name VARCHAR(50)
);
```

b. Foreign Key

- Establishes a relationship between two tables.

sql

```sql
CREATE TABLE Departments (
  id INT PRIMARY KEY,
  department_name VARCHAR(50)
);

CREATE TABLE Employees (
  id INT,
  name VARCHAR(50),
  department_id INT,
  FOREIGN KEY (department_id) REFERENCES Departments(id)
);
```

c. Unique

- Ensures all values in a column are unique.

sql

```sql
CREATE TABLE Employees (
  id INT,
  email VARCHAR(50) UNIQUE
);
```

d. Check

- Enforces a condition on values in a column.

sql

```sql
CREATE TABLE Employees (
  id INT,
```

```
  age INT CHECK (age >= 18)
);
```

e. Not Null

- Ensures a column cannot have a NULL value.

```sql
CREATE TABLE Employees (
  id INT,
  name VARCHAR(50) NOT NULL
);
```

---

8. Transactions in SQL

- A **transaction** is a sequence of database operations that forms a single logical unit of work.

a. COMMIT

- Saves all changes made during the transaction.

b. ROLLBACK

- Undoes changes made in the transaction.

Example:

```sql
BEGIN TRANSACTION;

UPDATE Employees SET salary = salary * 1.1 WHERE department = 'HR';

ROLLBACK; -- Undoes the salary update

COMMIT; -- Saves changes
```

---

9. Views

- **Views** are virtual tables created from the result set of a SQL query.
  - **Creating a View**:

```sql
CREATE VIEW HR_Employees AS
SELECT name, salary FROM Employees WHERE department = 'HR';
```

  - **Using a View**:

```sql
SELECT * FROM HR_Employees;
```

---

10. Advanced SQL: Window Functions

- **Window functions** perform calculations across rows related to the current row.

a. ROW_NUMBER

- Assigns a unique number to each row.

```sql
SELECT name, salary, ROW_NUMBER() OVER(ORDER BY salary DESC) AS row_num
FROM Employees;
```

b. RANK

- Assigns a rank to each row within a partition.

```sql
SELECT name, salary, RANK() OVER(ORDER BY salary DESC) AS rank
FROM Employees;
```

c. PARTITION BY

- Divides result into partitions to perform calculations on each partition.

```sql
SELECT name, department, salary,
       RANK() OVER(PARTITION BY department ORDER BY salary DESC) AS rank
FROM Employees;
```

---

11. Normalization

- **Normalization** organizes database tables to minimize redundancy and dependency.
  - **1NF**: Each column should contain atomic values.
  - **2NF**: Remove partial dependencies.
  - **3NF**: Remove transitive dependencies.

---

12. Common Interview Questions

1. **What are joins and types of joins in SQL?**
2. **Explain normalization and its types.**
3. **What are primary and foreign keys?**
4. **Difference between `WHERE` and `HAVING` clauses?**
5. **How would you delete duplicate rows from a table?**
6. **Explain the use of indexes.**

These concepts cover a strong foundational knowledge of SQL for interviews, enabling you to confidently answer most entry-level SQL questions.