Basic Concepts

Q1. What is a database?

A database is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are managed by Database Management Systems (DBMS).

Follow-up Question: What are some examples of databases?

Answer: Examples include MySQL, PostgreSQL, Oracle, MongoDB, and Microsoft SQL Server.

Q2. What is DBMS?

A Database Management System (DBMS) is software that interacts with end-users, applications, and the database itself to capture and analyze data. The DBMS provides administrative operations, such as change management, performance monitoring/tuning, and backup and recovery.

Follow-up Question: What are some common features of DBMS?

Answer: Common features include data storage, retrieval, update, user administration, and concurrency control.

Q3. What are advantages of DBMS over traditional file-based systems?

1. **Data Redundancy and Inconsistency**: DBMS controls data redundancy by integrating all the data into a single database and thus reduces inconsistency.
2. **Data Sharing**: Multiple users can access and share data concurrently.
3. **Data Security**: DBMS provides security by allowing only authorized users to access the database.
4. **Data Integrity**: DBMS ensures data integrity by enforcing constraints on data.
5. **Backup and Recovery**: DBMS offers robust backup and recovery options.

Follow-up Question: What is an example of data inconsistency?

Answer: If customer information is stored in multiple files and one file is updated while the others are not, this creates inconsistency.

Q4. What are the various types of DBMS? Give examples of each.

1. **Hierarchical DBMS**: Data is organized into a tree-like structure. Example: IBM Information Management System (IMS).
2. **Network DBMS**: Data is organized in a graph, allowing many-to-many relationships. Example: Integrated Data Store (IDS).
3. **Relational DBMS (RDBMS)**: Data is organized in tables. Example: MySQL, PostgreSQL, Oracle.

4. **Object-oriented DBMS (OODBMS)**: Data is stored as objects, similar to object-oriented programming. Example: db4o, ObjectDB.
5. **NoSQL DBMS**: Used for large-scale data storage and for real-time web applications. Example: MongoDB, Cassandra.

Follow-up Question: What is the primary difference between relational and object-oriented databases?

Answer: Relational databases store data in tables and use SQL for queries, while object-oriented databases store data as objects, similar to programming languages like Java or C++.

Q5. What is the difference between DBMS and RDBMS? Give examples of RDBMS.

DBMS is a general term for a system that manages databases. RDBMS is a specific type of DBMS based on the relational model introduced by E.F. Codd.

- **DBMS**: Manages databases, does not necessarily use a tabular structure. Examples: File system, XML databases.
- **RDBMS**: Manages relational databases, uses tables with rows and columns. Examples: MySQL, PostgreSQL, Oracle, SQL Server.

Follow-up Question: What is a relational model?

Answer: A relational model is a way to structure and query data using relations (tables) with tuples (rows) and attributes (columns).

Keys and Constraints

Q6. What are super, primary, candidate, and foreign keys? Explain with examples. Also, what is the difference between primary and unique key? And difference between foreign key and primary key.

1. **Super Key**: A set of one or more columns that uniquely identifies a row in a table.
    - Example: {StudentID, Email},
2. **Candidate Key**: A minimal super key, i.e., a super key with no redundant attributes.
    - Example: ,
3. **Primary Key**: A candidate key chosen to uniquely identify rows in a table. It cannot be NULL.
    - Example: StudentID
4. **Foreign Key**: A column that creates a relationship between two tables. It references the primary key of another table.
    - Example: CourseID in Enrollments table referencing CourseID in Courses table.

Difference between Primary Key and Unique Key:

- **Primary Key**: Uniquely identifies a row and cannot be NULL.
- **Unique Key**: Uniquely identifies a row but can have one NULL value.

Difference between Foreign Key and Primary Key:

- **Primary Key**: Uniquely identifies a row within its table.
- **Foreign Key**: Establishes a relationship between rows in two tables.

Follow-up Question: Why is a primary key important?

Answer: It ensures each row is uniquely identifiable and enforces entity integrity.

Database Normalization

Q7. What is database normalization?

Normalization is the process of organizing data to minimize redundancy and improve data integrity. It involves dividing large tables into smaller, related tables and defining relationships between them.

Follow-up Question: Why is normalization important?

Answer: It reduces data redundancy, ensures data integrity, and improves query performance.

Q8. What are the various Normal Forms? Explain the use of all normal forms till BCNF with short examples.

1. **First Normal Form (1NF)**: Ensures that the table has only atomic (indivisible) values.
    - Example: A table with a column "PhoneNumbers" containing a single phone number per row.
2. **Second Normal Form (2NF)**: Meets all the requirements of 1NF and all non-key attributes are fully functional dependent on the primary key.
    - Example: Splitting a table with columns (StudentID, CourseID, Grade) into two tables, one with (StudentID, CourseID) and another with (CourseID, Grade).
3. **Third Normal Form (3NF)**: Meets all the requirements of 2NF and all non-key attributes are non-transitively dependent on the primary key.
    - Example: Removing columns like "InstructorName" from a table that only needs "InstructorID" and "CourseID".
4. **Boyce-Codd Normal Form (BCNF)**: Meets all the requirements of 3NF and for every functional dependency (X → Y), X is a super key.
    - Example: Ensuring that all functional dependencies have a super key as their determinant.

Follow-up Question: Can you give an example of a table in 2NF but not in 3NF?

Answer: A table with columns (StudentID, CourseID, InstructorID, InstructorName) where InstructorName is dependent on InstructorID, not on (StudentID, CourseID).

Q9. Differentiate between 2NF and 3NF with example. And also between 3NF and BCNF.

- **2NF vs. 3NF**:
    - 2NF: Removes partial dependencies.
    - 3NF: Removes transitive dependencies.
    - Example: If a table (StudentID, CourseID, InstructorName) is in 2NF but not in 3NF because InstructorName depends on InstructorID, which is not a primary key.
- **3NF vs. BCNF**:
    - 3NF: Removes transitive dependencies.
    - BCNF: Ensures that every determinant is a super key.
    - Example: A table (CourseID, InstructorID, CourseTime) where InstructorID → CourseTime but InstructorID is not a super key.

Follow-up Question: What is a transitive dependency?

Answer: A transitive dependency is when a non-key attribute depends on another non-key attribute.

Q10. What is data redundancy and how does normalization reduce that?

Data redundancy occurs when the same piece of data is stored in multiple places. Normalization reduces redundancy by ensuring that data is stored in only one place and referenced elsewhere, preventing duplicate data entries.

Follow-up Question: What is an example of data redundancy?

Answer: Storing a student's address in multiple tables, which can lead to inconsistencies if one instance is updated but others are not.

Q11. What is the relationship between Normal Forms and partial dependency and transitive dependency?

- **1NF**: Eliminates repeating groups and ensures atomicity.
- **2NF**: Eliminates partial dependencies (a non-key attribute depends on part of a composite key).
- **3NF**: Eliminates transitive dependencies (a non-key attribute depends on another non-key attribute).
- **BCNF**: Ensures that every determinant is a super key.

Follow-up Question: Can a table be in BCNF but not in 3NF?

Answer: No, if a table is in BCNF, it is also in 3NF because BCNF is a stricter form of 3NF.

SQL Concepts

Q12. What is SQL?

SQL (Structured Query Language) is a standardized programming language used to manage and manipulate relational databases.

Follow-up Question: What are some common SQL operations?

Answer: Common operations include SELECT, INSERT, UPDATE, DELETE, and JOIN.

Q13. What is DDL, DML, and DCL in SQL?

- **DDL (Data Definition Language)**: Defines the structure of the database. Commands: CREATE, ALTER, DROP.
- **DML (Data Manipulation Language)**: Manipulates data stored in the database

. Commands: SELECT, INSERT, UPDATE, DELETE.

- **DCL (Data Control Language)**: Controls access to data in the database. Commands: GRANT, REVOKE.

Follow-up Question: What does the ALTER command do?

Answer: ALTER modifies the structure of an existing database object, such as a table.

Q14. What is the difference between TRUNCATE, DELETE, and DROP commands?

- **TRUNCATE**: Removes all rows from a table without logging individual row deletions. Cannot be rolled back.
- **DELETE**: Removes specific rows based on a condition. Can be rolled back.
- **DROP**: Deletes the entire table structure along with its data. Cannot be rolled back.

Follow-up Question: Which command would you use to remove all rows from a table but keep the table structure?

Answer: TRUNCATE.

Q15. What is the difference between HAVING and WHERE clause?

- **WHERE**: Filters rows before any groupings are made.
- **HAVING**: Filters groups after the GROUP BY clause has been applied.

Example:

```
SELECT Department, COUNT(*)
FROM Employees
WHERE Salary > 50000
GROUP BY Department
HAVING COUNT(*) > 10;
```

Follow-up Question: Can HAVING be used without GROUP BY?

Answer: No, HAVING is used to filter groups, so it requires GROUP BY.

Q16. What are the various aggregate operators in SQL? Explain with short examples.

1. **COUNT()**: Returns the number of rows.
    - Example: `SELECT COUNT(*) FROM Employees;`

2. **SUM()**: Returns the total sum of a numeric column.
    - Example: `SELECT SUM(Salary) FROM Employees;`
3. **AVG()**: Returns the average value of a numeric column.
    - Example: `SELECT AVG(Salary) FROM Employees;`
4. **MAX()**: Returns the maximum value in a column.
    - Example: `SELECT MAX(Salary) FROM Employees;`
5. **MIN()**: Returns the minimum value in a column.
    - Example: `SELECT MIN(Salary) FROM Employees;`

Follow-up Question: How would you find the highest salary in each department?

Answer:

```
SELECT Department, MAX(Salary)
FROM Employees
GROUP BY Department;
```

Q17. What is GROUP BY and ORDER BY and the difference between them?

- **GROUP BY**: Groups rows sharing a property so aggregate functions can be applied.
    - Example: `SELECT Department, COUNT(*) FROM Employees GROUP BY Department;`
- **ORDER BY**: Sorts the result set by specified columns.
    - Example: `SELECT * FROM Employees ORDER BY Salary DESC;`

Difference:

- **GROUP BY**: Used for aggregation.
- **ORDER BY**: Used for sorting results.

Follow-up Question: Can you use ORDER BY with GROUP BY?

Answer: Yes, to sort the grouped results.

Q18. What are various kinds of joins? Give examples of each.

1. **INNER JOIN**: Returns records that have matching values in both tables.
    - Example:

```
SELECT Employees.Name, Departments.Name
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.ID;
```

2. **LEFT JOIN**: Returns all records from the left table, and the matched records from the right table.

- Example:

```
SELECT Employees.Name, Departments.Name
FROM Employees
LEFT JOIN Departments ON Employees.DepartmentID = Departments.ID;
```

3. **RIGHT JOIN**: Returns all records from the right table, and the matched records from the left table.
   - Example:

```
SELECT Employees.Name, Departments.Name
FROM Employees
RIGHT JOIN Departments ON Employees.DepartmentID = Departments.ID;
```

4. **FULL OUTER JOIN**: Returns all records when there is a match in either left or right table.
   - Example:

```
SELECT Employees.Name, Departments.Name
FROM Employees
FULL OUTER JOIN Departments ON Employees.DepartmentID = Departments.ID;
```

Follow-up Question: When would you use a FULL OUTER JOIN?

Answer: When you need to include all records from both tables, regardless of matches.

Q19. What is the difference between self join and cross join? Show with example.

- **Self Join**: A table is joined with itself.
  - Example:

```
SELECT A.Name AS Employee1, B.Name AS Employee2
FROM Employees A, Employees B
WHERE A.ManagerID = B.ID;
```

- **Cross Join**: Returns the Cartesian product of the two tables.
  - Example:

```
SELECT Employees.Name, Departments.Name
FROM Employees
CROSS JOIN Departments;
```

Follow-up Question: What is a Cartesian product?

Answer: The result of a CROSS JOIN, which is a set of all possible combinations of rows from two tables.

Q20. What is the difference between inner join and outer join? Explain various kinds of outer joins with real-life example tables.

- **Inner Join**: Returns only the rows with matching values in both tables.
    - Example: Joining Employees and Departments to list employees with their department names.
- **Outer Join**: Returns matched rows and unmatched rows from one or both tables.
    1. **Left Outer Join**: Returns all rows from the left table and matched rows from the right table.
        - Example: List all employees and their departments, including employees without a department.

```
SELECT Employees.Name, Departments.Name
FROM Employees
LEFT JOIN Departments ON Employees.DepartmentID = Departments.ID;
```

    2. **Right Outer Join**: Returns all rows from the right table and matched rows from the left table.
        - Example: List all departments and their employees, including departments without employees.

```
SELECT Employees.Name, Departments.Name
FROM Employees
RIGHT JOIN Departments ON Employees.DepartmentID = Departments.ID;
```

    3. **Full Outer Join**: Returns all rows when there is a match in either left or right table.
        - Example: List all employees and departments, including those without matches.

```
SELECT Employees.Name, Departments.Name
FROM Employees
FULL OUTER JOIN Departments ON Employees.DepartmentID = Departments.ID;
```

Follow-up Question: Which join would you use to find employees without a department?

Answer: LEFT JOIN with a WHERE clause to filter NULL department values.

```
SELECT Employees.Name
FROM Employees
LEFT JOIN Departments ON Employees.DepartmentID = Departments.ID
WHERE Departments.ID IS NULL;
```

ACID Properties

Q21. Explain the ACID properties with examples.

1. **Atomicity**: Ensures that all operations in a transaction are completed; if not, the transaction is aborted.
    - Example: Transferring money between accounts should either fully succeed or not at all.
2. **Consistency**: Ensures that a transaction brings the database from one valid state to another.
    - Example: Ensuring that after a transfer, the total amount in both accounts remains the same.
3. **Isolation**: Ensures that transactions are executed in isolation from one another.
    - Example: Concurrent transactions should not interfere with each other.
4. **Durability**: Ensures that once a transaction is committed, it remains so, even in the event of a system failure.
    - Example: After completing a transfer, the changes should persist even if the system crashes.

Follow-up Question: What happens if a transaction fails during execution?

Answer: If a transaction fails, atomicity ensures that all previous operations are rolled back, leaving the database in its initial state.

Frequently Asked SQL Queries

Q22. Give 5 most frequently asked queries with solutions in interviews for a fresher.

1. **Find all employees with salary greater than 50000.**

```
SELECT * FROM Employees WHERE Salary > 50000;
```

2. **List the number of employees in each department.**

```
SELECT DepartmentID, COUNT(*) AS EmployeeCount
FROM Employees
GROUP BY DepartmentID;
```

3. **Retrieve employee names and their respective department names.**

```
SELECT Employees.Name, Departments.Name AS DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.ID;
```

4. **Find the second highest salary in the employees table.**

```
SELECT MAX(Salary) AS SecondHighestSalary
FROM Employees
WHERE Salary < (SELECT MAX(Salary) FROM Employees);
```

5. **Delete duplicate rows from a table based on a specific column.**

```
DELETE FROM Employees
WHERE ID NOT IN (
    SELECT MIN(ID)
    FROM Employees
    GROUP BY Email
);
```

Follow-up Question: How would you handle NULL values in a query?

Answer: Use IS NULL or IS NOT NULL in the WHERE clause.

```
SELECT * FROM Employees WHERE DepartmentID IS NULL;
```

Certainly! Here are the SQL queries to retrieve the person with the highest salary and the person with the 5th highest salary:

Query to get the person with the highest salary:

```
SELECT Name, Salary
FROM Employees
WHERE Salary = (SELECT MAX(Salary) FROM Employees);
```

Query to get the person with the 5th highest salary:

To get the 5th highest salary, we can use a subquery with the `DISTINCT` clause and `ORDER BY` to sort the salaries, then use the `OFFSET` and `LIMIT` clauses.

```
SELECT Name, Salary
FROM Employees
WHERE Salary = (
    SELECT DISTINCT Salary
    FROM Employees
    ORDER BY Salary DESC
    OFFSET 4 ROWS
    FETCH NEXT 1 ROW ONLY
);
```

Explanation:

1. **Highest Salary**:
    - The subquery `SELECT MAX(Salary) FROM Employees` finds the maximum salary in the `Employees` table.
    - The outer query selects the `Name` and `Salary` of the employee(s) with that maximum salary.
2. **5th Highest Salary**:
    - The subquery `SELECT DISTINCT Salary FROM Employees ORDER BY Salary DESC` orders all distinct salaries in descending order.
    - The `OFFSET 4 ROWS` skips the first 4 highest salaries.
    - The `FETCH NEXT 1 ROW ONLY` retrieves the 5th highest salary.
    - The outer query then selects the `Name` and `Salary` of the employee(s) with that salary.

These queries assume the `Employees` table has columns `Name` and `Salary`. Adjust the column names if necessary to match your actual table structure. This guide should provide a comprehensive foundation for a fresher preparing for an interview in databases and SQL.