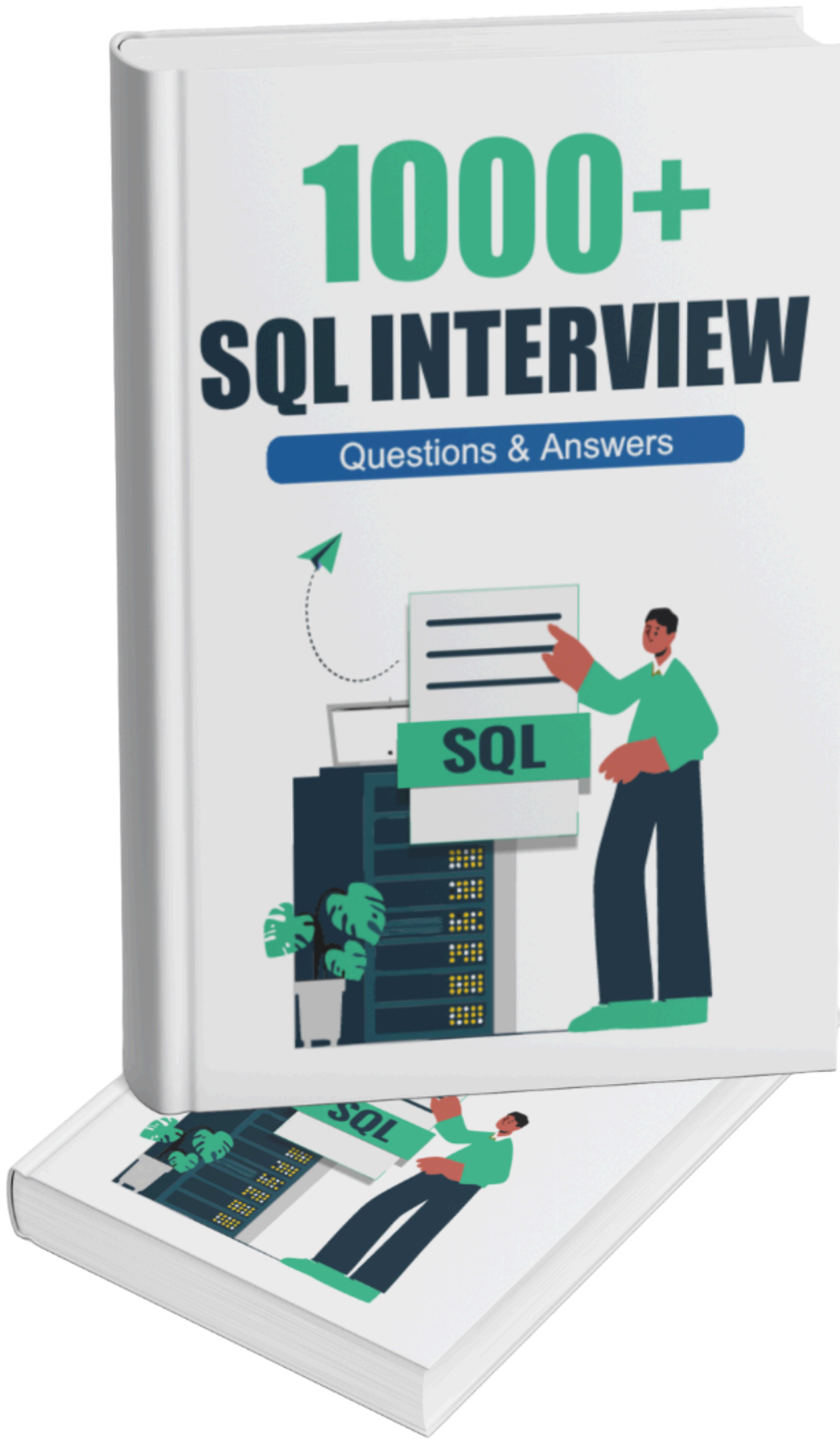

coding_knowledge
harry


1000+
SQL INTERVIEW
Questions & Answers
SQL

coding_knowledge
harry

- INNER, LEFT, RIGHT, FULL OUTER, SELF, and CROSS JOINs
- **Subqueries:**
  - Single-row, Multi-row, and Correlated Subqueries
- **Set Operations:**
  - UNION, UNION ALL, INTERSECT, EXCEPT
- **Common Table Expressions (CTEs):**
  - Non-recursive and Recursive CTEs

- **Schema Design:**
  - Designing effective database schemas
- **Normalization:**
  - 1NF, 2NF, 3NF, BCNF
- **Denormalization:**
  - Trade-offs for performance

- **Primary vs Unique Indexes:**
  - Differences and usage
- **Clustered vs Non-clustered Indexes:**
  - Performance impact and differences

- **Query Optimization Techniques:**
  - Using EXPLAIN or EXPLAIN PLAN
  - Tuning complex queries
- **Partitioning Data:**
  - Horizontal and Vertical Partitioning

- **SQL vs NoSQL Databases:**
  - Key differences and when to use
- **Basic NoSQL Concepts:**
  - Key-Value stores, Document stores, Column-family stores

- **Cloud SQL Platforms:**
  - AWS RDS, Azure SQL Database, Google Cloud SQL
- **Managing SQL Databases on Cloud:**
  - Backup, scaling, and performance tuning

Chapter 1: SQL Basics

THEORETICAL QUESTIONS

1. What is SQL, and why is it important?

Answer:
SQL (Structured Query Language) is a standard language designed to manage and manipulate relational databases. It plays a vital role in building and maintaining database-driven applications by offering a clear way to query, update, and control data stored in tables. SQL is highly adaptable and works with different relational database management systems like MySQL, PostgreSQL, Microsoft SQL Server, and Oracle.

The importance of SQL lies in its **declarative nature**: instead of specifying *how* the data should be retrieved or manipulated, you simply declare *what* you need. SQL takes care of the rest. SQL is used in various domains such as web development, data analytics, and enterprise resource management, ensuring structured data handling.

For Example:

```sql
-- Retrieving all data from the employees table.
SELECT * FROM employees;
```

This query retrieves all columns and rows from the `employees` table. This ability to extract structured data efficiently makes SQL indispensable in software applications.

2. What is the purpose of the SELECT statement in SQL?

Answer:
The `SELECT` statement retrieves data from one or more tables. It forms the

foundation of most SQL queries because it enables users to query the database and fetch only the needed data. It can also apply filters with the `WHERE` clause, group data with `GROUP BY`, sort data with `ORDER BY`, and perform calculations with **aggregate functions**.

The `SELECT` statement is highly versatile and supports the use of aliases, joins, and subqueries for complex queries.

For Example:

```sql
-- Selecting specific columns based on a condition.
SELECT first_name, last_name, department
FROM employees
WHERE department = 'HR';
```

This query retrieves only the first and last names of employees working in the HR department. The `WHERE` clause acts as a filter, ensuring only relevant data is displayed.

3. How is the INSERT statement used in SQL?

Answer:
The `INSERT` statement allows you to add new records to a table. It is used when new data needs to be added to the database, such as when a new employee joins a company or a new product is added to an inventory.

The values provided during insertion must match the **column data types** and constraints like `NOT NULL` or `UNIQUE`. You can insert values into specific columns or all columns if you provide values for every field.

For Example:

```sql
-- Adding a new record to the employees table.
INSERT INTO employees (first_name, last_name, department, salary)
VALUES ('John', 'Doe', 'Finance', 60000);
```

This query adds a new employee named John Doe to the Finance department with a salary of 60,000. Without the `department` and `salary` columns, the insertion would fail if those columns are marked as `NOT NULL`.

4. What is the role of the UPDATE statement?

Answer:

The `UPDATE` statement modifies existing records in a table. It's useful when you need to **alter data dynamically**, such as changing an employee's salary after a promotion or correcting an error in the data. The `SET` keyword identifies the column(s) to update, and the `WHERE` clause ensures that only the relevant records are affected.

Best Practice: Always include a `WHERE` **clause** to avoid updating all rows by mistake.

For Example:

```sql
-- Updating the salary for a specific employee.
UPDATE employees
SET salary = 65000
WHERE first_name = 'John' AND last_name = 'Doe';
```

In this query, only the salary of John Doe is updated. Without the `WHERE` clause, the salary of every employee would change to 65,000.

---

5. How is the DELETE statement used in SQL?

Answer:
The `DELETE` statement removes data from a table. This operation is permanent, meaning that the deleted records are not recoverable unless there is a backup. The `WHERE` clause specifies which records should be deleted. **Using DELETE without a `WHERE` clause deletes all rows in the table.**

Best Practice: Always include a `WHERE` **clause** unless you are intentionally clearing the entire table.

For Example:

```sql
-- Deleting an employee record.
DELETE FROM employees
WHERE first_name = 'John' AND last_name = 'Doe';
```

This query removes the record of John Doe from the `employees` table. If you omit the `WHERE` clause, all employee records will be deleted.

6. What are the different data types in SQL?

Answer:
SQL supports several **data types** to manage different forms of data efficiently. Proper data types ensure **data integrity** and optimize storage. Here are the common data types:

- **Numeric types:**
  - `INT`: Stores whole numbers.
  - `DECIMAL(p, s)`: Stores numbers with precision, ideal for financial data.
  - `FLOAT`: Stores floating-point numbers.
- **String types:**
  - `CHAR(n)`: Fixed-length string.
  - `VARCHAR(n)`: Variable-length string.
  - `TEXT`: Large strings.

- Date/Time types:
  - `DATE`: Stores date values (YYYY-MM-DD).
  - `TIME`: Stores time (HH:MM
).
  - `TIMESTAMP`: Stores date and time values.
- Boolean type:
  - `BOOLEAN`: Stores `TRUE` or `FALSE` values.

For Example:

```sql
-- Creating a table with various data types.
CREATE TABLE employees (
    id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    salary DECIMAL(10, 2),
    join_date DATE,
    is_active BOOLEAN
);
```

This example defines an `employees` table using appropriate data types to match the type of data each column will store.

7. How are arithmetic operators used in SQL?

Answer:
SQL supports arithmetic operations, including **addition (+), subtraction (-), multiplication (*), and division (/)**. These operators are often used within `SELECT` statements to perform **calculations** on numerical data, such as calculating totals, averages, or percentages.

For Example:

```sql
-- Calculating a 10% bonus for each employee.
SELECT first_name, last_name, salary, salary * 0.10 AS bonus
FROM employees;
```

This query calculates a 10% bonus on the employees' salaries and displays it along with their names. Such operations are often used to derive insights directly from the data.

8. What are comparison operators in SQL, and how are they used?

Answer:
Comparison operators allow SQL queries to compare values within conditions, typically in the `WHERE` clause. The main operators are:

- `=`: Equal to.
- `!=` or `<>`: Not equal to.
- `<`: Less than.
- `>`: Greater than.
- `<=`: Less than or equal to.
- `>=`: Greater than or equal to.

For Example:

```sql
-- Retrieving employees with a salary greater than 50,000.
SELECT first_name, last_name, salary
FROM employees
WHERE salary > 50000;
```

This query returns employees with a salary greater than 50,000. The `>` operator filters out the employees who do not meet the salary condition.

9. Explain the use of logical operators in SQL.

Answer:

SQL offers **logical operators** like `AND`, `OR`, and `NOT` to **combine multiple conditions** in a query. These operators are particularly useful when filtering data based on multiple criteria.

- `AND`: Both conditions must be true.
- `OR`: At least one condition must be true.
- `NOT`: Negates a condition.

For Example:

```sql
-- Selecting employees with complex conditions.
SELECT first_name, last_name
FROM employees
WHERE department = 'Finance' AND salary > 50000;
```

This query uses `AND` to filter employees working in Finance with salaries above 50,000.

10. How can the DATE data type be used in SQL queries?

Answer:

The `DATE` data type stores only the **year, month, and day**. It allows you to compare dates and perform operations such as filtering records based on a date range or identifying recent entries. SQL also supports functions like `CURDATE()` to work with the current date.

For Example:

```sql
-- Finding employees who joined after January 1, 2023.
SELECT first_name, last_name, join_date
FROM employees
WHERE join_date > '2023-01-01';
```

This query returns employees who joined after January 1, 2023, using the `>` operator on the `join_date` column.

11. What is the difference between CHAR and VARCHAR data types in SQL?

Answer:

Both `CHAR` and `VARCHAR` store text values, but they handle storage differently:

- `CHAR(n)`: A fixed-length string. If the input length is less than the defined size `n`, it pads the remaining space with **extra spaces**. This makes it efficient when all entries are of the same length, like country codes or status codes.
- `VARCHAR(n)`: A variable-length string. It stores only the input size plus **one extra byte** for tracking the length. It saves space when values vary in size, but accessing `VARCHAR` values may be slightly slower due to dynamic size management.

For Example:

```sql
-- Demonstrating CHAR and VARCHAR behavior.
CREATE TABLE test (
    fixed_length CHAR(5),
    variable_length VARCHAR(5)
);
```

```
INSERT INTO test (fixed_length, variable_length)
VALUES ('abc', 'abc');
```

In this case, the `fixed_length` value will be stored as `'abc  '` (with two trailing spaces), while the `variable_length` value will be stored as `'abc'`. This difference affects how strings are stored and compared.

12. What is the purpose of the DISTINCT keyword in SQL?

Answer:

The `DISTINCT` keyword ensures that the query returns only unique (non-duplicate) values. It is helpful when a column contains duplicate values, and you only want to see unique entries. Without `DISTINCT`, the query would return every instance of a value, including duplicates.

For Example:

```
-- Retrieving distinct departments.
SELECT DISTINCT department
FROM employees;
```

If multiple employees belong to the same department, the query will return only one entry for each department. This is useful when you want to find the unique categories, such as departments or job titles, from large datasets.

13. What is the purpose of the LIMIT keyword in SQL?

Answer:

The `LIMIT` clause restricts the number of rows returned by a query. It is often used

when **paging through large datasets** (for example, showing 10 rows at a time) or when you need only a subset of results, such as the top `n` records.

For Example:

```sql
-- Retrieving the top 3 highest-paid employees.
SELECT first_name, last_name, salary
FROM employees
ORDER BY salary DESC
LIMIT 3;
```

This query returns only the top 3 employees with the highest salaries. Without `LIMIT`, the query would return all employees sorted by salary.

14. What is a PRIMARY KEY in SQL?

Answer:
A `PRIMARY KEY` is a constraint that ensures each row in a table is **uniquely identifiable**. It guarantees two properties:

1. **Uniqueness:** No two rows can have the same value for the primary key column(s).
2. **Non-nullability:** A primary key column cannot contain `NULL` values.

Each table can have only **one primary key**, and it can consist of one or more columns (in which case it is called a **composite key**).

For Example:

```sql
-- Creating a table with a primary key.
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
```

```
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);
```

Here, `employee_id` ensures that each employee has a unique identifier.

15. What is a FOREIGN KEY in SQL?

Answer:

A `FOREIGN KEY` establishes a link between two tables by referencing the **primary key** of another table. It enforces **referential integrity**, ensuring that the value in the foreign key column must exist in the referenced table. If the referenced value is deleted or updated, SQL can prevent, cascade, or restrict the change to maintain data integrity.

For Example:

```
-- Creating a foreign key relationship.
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50)
);

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

This ensures that every `department_id` in the `employees` table matches an existing `department_id` in the `departments` table.

16. How does the WHERE clause work in SQL?

Answer:

The `WHERE` clause is used to **filter rows** based on specified conditions. It allows you to retrieve only the data that meets specific criteria, making queries more focused and efficient. Without `WHERE`, all rows in the table are returned. You can use comparison operators (`=`, `<`, `>`) and logical operators (`AND`, `OR`) within `WHERE`.

For Example:

```sql
-- Retrieving employees from the Finance department.
SELECT first_name, last_name
FROM employees
WHERE department = 'Finance';
```

This query returns only those employees who belong to the Finance department.

17. What is the purpose of the GROUP BY clause in SQL?

Answer:

The `GROUP BY` clause groups rows with the same values into **categories**. It is usually paired with **aggregate functions** like `SUM()`, `COUNT()`, or `AVG()` to calculate metrics for each group. For example, it can help find the total salary per department or the number of employees in each role.

For Example:

```sql
-- Grouping employees by department and calculating total salary.
SELECT department, SUM(salary) AS total_salary
FROM employees
GROUP BY department;
```


coding_knowledge
harry

A few of the final pages of the ebook 👉

```
location=global
```

Table Example:
A `KeyRotationLog` table tracks key rotations:

KeyID	RotationTime	Status
K-001	2024-10-22 15:00:00	Success

This ensures encryption keys are updated regularly.

78.

Scenario:
Your AWS RDS PostgreSQL instance is running out of storage, and you need to increase capacity without downtime.

Question:
How can you enable **storage autoscaling** in AWS RDS PostgreSQL?

Answer:
AWS RDS allows **storage autoscaling**, which increases storage capacity automatically as needed, ensuring continuous operation without manual intervention.

For Example:
Enable storage autoscaling:

```
aws rds modify-db-instance \
    --db-instance-identifier mydb \
    --max-allocated-storage 1000
```

Table Example:
A `StorageUsage` table tracks storage allocation:

InstanceID	AllocatedStorage	MaxStorage	Status
mydb	950 GB	1000 GB	Scaling

This ensures uninterrupted operation as data grows.

79.

Scenario:
Your Azure SQL Database needs to support **geo-replication** to ensure data availability across regions.

Question:
How do you configure **geo-replication** in Azure SQL?

Answer:
Azure SQL supports **geo-replication**, which replicates data asynchronously to other regions. This ensures high availability and disaster recovery in case of regional outages.

For Example:
Enable geo-replication:

```
ALTER DATABASE mydb ADD SECONDARY ON SERVER myserver2;
```

Table Example:
A `GeoReplication` table tracks replica status:

ReplicaID	PrimaryRegion	ReplicaRegion	Status
G-001	East US	West US	Active

This ensures the database remains available across regions.

80.

Scenario:
Your Google Cloud SQL instance serves multiple applications, and you need to monitor **query execution times** to optimize performance.

Question:
How can you track and optimize query execution times in Google Cloud SQL?

Answer:
Use **Cloud SQL Insights** to monitor query execution times and identify slow queries. Indexing, query rewriting, and resource scaling are common optimization techniques.

For Example:
Run a query to view execution statistics:

```
SELECT query_id, total_time / calls AS avg_time FROM pg_stat_statements
ORDER BY avg_time DESC;
```

Table Example:
A `QueryPerformance` table helps track slow queries:

QueryID	AvgExecutionTime (ms)	LastExecutedTime	Status

Q-001	1500	2024-10-22 14:00:00	Tuned

This ensures continuous performance monitoring and optimization.