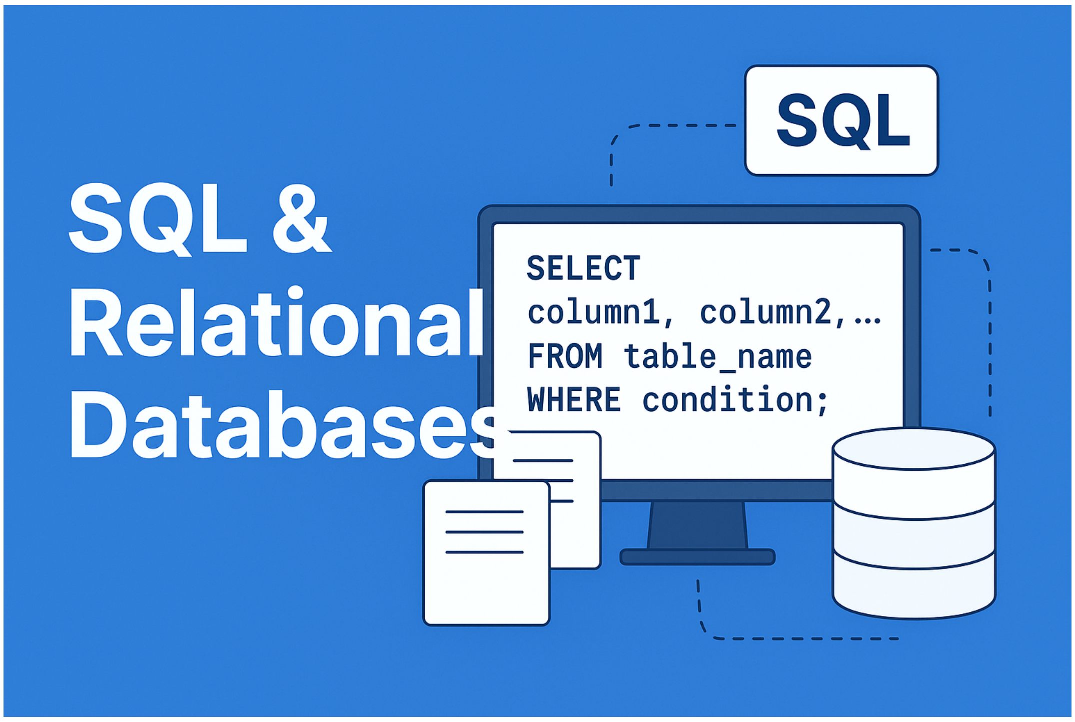
SQL & Relational Databases Explained: Basics to Advanced with Examples



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

Databases are the backbone of almost every modern application. Whether you are using a banking system, an e-commerce website, a
social media platform, or even a student portal like The Cubicals, all of them rely on databases to store, organize, and manage
information effectively.

WHAT IS A DATABASE?

A database is a structured collection of data that can be stored, retrieved, and managed electronically. Instead of keeping files
scattered across the system, databases allow us to organize data in a systematic way so that it can be used efficiently.

* Example 1: A college database may store student records, courses, faculty details, and exam results.
* Example 2: An e-commerce database stores product details, customer information, and orders.

TYPES OF DATABASES

Over time, different types of databases have been developed. The most common are:

* Relational Databases: Data is stored in tables with rows and columns.
* NoSQL Databases: Focus on flexibility, storing data as documents, key-value pairs, or graphs.
* Distributed Databases: Data is stored across multiple locations.
* Cloud Databases: Hosted on cloud platforms like AWS, Azure, and Google Cloud.

WHAT IS A RELATIONAL DATABASE?

A relational database organizes data into tables (also called relations). Each table consists of rows (records) and columns
(attributes). Relationships between different tables are defined using keys. This model was first introduced by Edgar F. Codd in
1970 and has since become the most widely used database model.

KEY FEATURES OF RELATIONAL DATABASES

* Data is organized into tables with rows and columns.
* Tables can be linked using primary keys and foreign keys.
* They support powerful query languages like SQL.

* Provide data integrity and consistency.

EXAMPLE OF A RELATIONAL DATABASE

Imagine we are creating a simple database for a library:

Table: Students

StudentID	Name	Class
1	Riya	10th
2	Arjun	11th

Table: Books

BookID	Title	Author
101	Database Management	Navathe
102	Learn SQL	Alan Smith

Here, students borrow books. The relationship can be established by creating another table called BorrowedBooks where StudentID
and BookID are linked.

WHAT IS SQL?

SQL (Structured Query Language) is the standard language used to communicate with relational databases. With SQL, you can create
tables, insert data, update records, delete data, and perform complex queries to fetch

information.

WHY SQL IS IMPORTANT?

* It allows developers to interact with databases using simple commands.
* Almost all relational databases (MySQL, PostgreSQL, Oracle, SQL Server) use SQL.
* It provides security, scalability, and data consistency.

BASIC SQL COMMANDS (CATEGORIES)

SQL commands are grouped into four main categories:

* DDL (Data Definition Language): Used to define structures (CREATE, ALTER, DROP).
* DML (Data Manipulation Language): Used to manipulate data (INSERT, UPDATE, DELETE).
* DQL (Data Query Language): Used to query data (SELECT).
* DCL (Data Control Language): Used for permissions (GRANT, REVOKE).

SQL SYNTAX AND STRUCTURE

SQL follows a very simple and human-readable syntax. Every SQL statement is made up of keywords, identifiers (like table names and
column names), and conditions. A statement usually ends with a semicolon (;).

BASIC SQL QUERY STRUCTURE

```
SELECT column1, column2, ...
```

```
FROM table_name
WHERE condition;
```

Example: To fetch names of all students studying in class 10:

```
SELECT Name
FROM Students
WHERE Class = '10th';
```

UNDERSTANDING KEYS IN RELATIONAL DATABASES

Keys are an essential part of relational databases. They help identify records uniquely and maintain relationships between
different tables.

1. PRIMARY KEY

A primary key is a column (or combination of columns) that uniquely identifies each record in a table. A table can have only one
primary key.

```
CREATE TABLE Students (
StudentID INT PRIMARY KEY,
Name VARCHAR(50),
Class VARCHAR(20)
);
```

Here, StudentID is the primary key because no two students can have the same ID.

2. FOREIGN KEY

A foreign key in one table refers to the primary key in another table. It helps establish relationships between tables.

```
CREATE TABLE BorrowedBooks (
BorrowID INT PRIMARY KEY,
StudentID INT,
BookID INT,
FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
FOREIGN KEY (BookID) REFERENCES Books(BookID)
);
```

Here, StudentID and BookID act as foreign keys referencing the Students and Books tables.

3. CANDIDATE KEY

A candidate key is any column (or set of columns) that can uniquely identify a row in a table. Every table may have multiple
candidate keys, but only one is chosen as the primary key.

Example: In the Students table, both StudentID and Email (if unique) could act as candidate keys.

4. COMPOSITE KEY

A composite key is formed by combining two or more columns to uniquely identify a row when a single column is not sufficient.

Example:

```
CREATE TABLE ClassSubjects (
Class VARCHAR(20),
Subject VARCHAR(50),
PRIMARY KEY (Class, Subject)
);
```

Here, neither Class alone nor Subject alone can uniquely identify a record, but the combination does.

RELATIONSHIPS IN RELATIONAL DATABASES

Relational databases allow linking tables together using keys. These links are called relationships. There are three main types:

1. ONE-TO-ONE RELATIONSHIP (1:1)

In a one-to-one relationship, each record in one table is related to exactly one record in another table.

Example: Each student has exactly one ID card.

Students(StudentID, Name, Class)
IDCards(CardID, StudentID, IssueDate)

2. ONE-TO-MANY RELATIONSHIP (1:M)

In this relationship, a record in one table can be related to multiple records in another table.

Example: A single student can borrow multiple books.

Students(StudentID, Name)
Books(BookID, Title)
BorrowedBooks(BorrowID, StudentID, BookID)

3. MANY-TO-MANY RELATIONSHIP (M:N)

In this type, multiple records in one table relate to multiple records in another table. It is usually implemented using a
junction table.

Example: Students can enroll in multiple courses, and each course can have multiple students.

Students(StudentID, Name)

Courses(CourseID, Title)

Enrollments(EnrollID, StudentID, CourseID)

SAMPLE SQL QUERIES WITH RELATIONSHIPS

Example 1: Fetch all books borrowed by a student with ID = 1

```
SELECT Books.Title
FROM BorrowedBooks
JOIN Books ON BorrowedBooks.BookID = Books.BookID
WHERE BorrowedBooks.StudentID = 1;
```

Example 2: Count how many students are enrolled in each course

```
SELECT Courses.Title, COUNT(Enrollments.StudentID) AS StudentCount
FROM Enrollments
JOIN Courses ON Enrollments.CourseID = Courses.CourseID
GROUP BY Courses.Title;
```

SQL CRUD OPERATIONS

CRUD stands for Create, Read, Update, and Delete. These are the four basic operations performed on database records. Let us look
at each operation with simple examples.

1. CREATE (INSERT DATA)

The INSERT statement is used to add new records into a table.

```
INSERT INTO Students (StudentID, Name, Class)
VALUES (1, 'Riya', '10th');
```

This command adds a new student to the Students table.

2. READ (FETCH DATA)

The SELECT statement is used to fetch data from tables.

```
SELECT * FROM Students;
```

This will display all records from the Students table.

3. UPDATE (MODIFY DATA)

The UPDATE statement is used to modify existing records.

```
UPDATE Students
SET Class = '11th'
WHERE StudentID = 1;
```

This changes Riya's class from 10th to 11th.

4. DELETE (REMOVE DATA)

The DELETE statement removes records from a table.

```
DELETE FROM Students
WHERE StudentID = 1;
```

This deletes the student with ID 1 from the table.

CONSTRAINTS IN SQL

Constraints are rules applied to table columns to maintain data accuracy and integrity.

1. NOT NULL

Ensures that a column cannot have NULL values.

```
CREATE TABLE Students (
StudentID INT PRIMARY KEY,
Name VARCHAR(50) NOT NULL
);
```

2. UNIQUE

Ensures that all values in a column are unique.

```
CREATE TABLE Users (
UserID INT PRIMARY KEY,
Email VARCHAR(100) UNIQUE
);
```

3. CHECK

Ensures that values in a column satisfy a specific condition.

```
CREATE TABLE Employees (
EmpID INT PRIMARY KEY,
Age INT CHECK (Age >= 18)
);
```

4. DEFAULT

Provides a default value for a column when no value is given.

```
CREATE TABLE Orders (
OrderID INT PRIMARY KEY,
Status VARCHAR(20) DEFAULT 'Pending'
);
```

JOINS IN SQL

A JOIN is used to combine records from two or more tables based on a related column. Let's explore the main types:

1. INNER JOIN

Returns only the matching records from both tables.

```
SELECT Students.Name, Books.Title
FROM BorrowedBooks
INNER JOIN Students ON BorrowedBooks.StudentID = Students.StudentID
INNER JOIN Books ON BorrowedBooks.BookID = Books.BookID;
```

2. LEFT JOIN

Returns all records from the left table, and matching records from the right table. Non-matching records from the right table will
show NULL.

```
SELECT Students.Name, Books.Title
FROM Students
LEFT JOIN BorrowedBooks ON Students.StudentID = BorrowedBooks.StudentID
LEFT JOIN Books ON BorrowedBooks.BookID = Books.BookID;
```

3. RIGHT JOIN

Returns all records from the right table, and matching records from the left table.

```
SELECT Students.Name, Books.Title
FROM Students
RIGHT JOIN BorrowedBooks ON Students.StudentID = BorrowedBooks.StudentID
RIGHT JOIN Books ON BorrowedBooks.BookID = Books.BookID;
```

4. FULL OUTER JOIN

Returns all records when there is a match in either left or right table. (Not supported in MySQL directly, but can be simulated
with UNION).

```
SELECT Students.Name, Books.Title
FROM Students
LEFT JOIN BorrowedBooks ON Students.StudentID = BorrowedBooks.StudentID
LEFT JOIN Books ON BorrowedBooks.BookID = Books.BookID
UNION
SELECT Students.Name, Books.Title
FROM Students
RIGHT JOIN BorrowedBooks ON Students.StudentID = BorrowedBooks.StudentID
RIGHT JOIN Books ON BorrowedBooks.BookID = Books.BookID;
```

REAL-WORLD EXAMPLE: STUDENT MANAGEMENT SYSTEM

Let's consider a mini-project style example: a Student Management System. Such a system could be used by platforms like The
Cubicals to manage student data, courses, and performance.

TABLES

```
Students (StudentID, Name, Email, Class)
Courses  (CourseID, Title, Teacher)
Enrollments (EnrollID, StudentID, CourseID, Grade)
```

QUERIES

1. Add a new student:

```
INSERT INTO Students (StudentID, Name, Email, Class)
VALUES (101, 'Saurabh Mishra', 'saurabh@thecubicals.online', 'B.Tech 2nd Year');
```

2. Show all students enrolled in a course:

```
SELECT Students.Name, Courses.Title
FROM Enrollments
JOIN Students ON Enrollments.StudentID = Students.StudentID
JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

3. Find average grade of a course:

```
SELECT Courses.Title, AVG(Enrollments.Grade) AS AvgGrade
FROM Enrollments
JOIN Courses ON Enrollments.CourseID = Courses.CourseID
GROUP BY Courses.Title;
```

ADVANCED SQL FEATURES

Apart from basic CRUD operations and joins, SQL provides advanced features that make databases powerful, efficient, and secure.

Let’s explore some of them.

1. VIEWS

A View is a virtual table created using a SQL query. It does not store data physically but provides a simplified way of accessing
complex queries.

```
CREATE VIEW StudentGrades AS
SELECT Students.Name, Courses.Title, Enrollments.Grade
FROM Enrollments
JOIN Students ON Enrollments.StudentID = Students.StudentID
JOIN Courses ON Enrollments.CourseID = Courses.CourseID;
```

Now, instead of writing the long query again, we can simply use:

```
SELECT * FROM StudentGrades;
```

2. INDEXES

Indexes improve the speed of data retrieval operations. They work like the index of a book, allowing faster lookups.

```
CREATE INDEX idx_student_name
```

```
ON Students(Name);
```

3. STORED PROCEDURES

Stored procedures are pre-written SQL statements stored in the database. They can be reused whenever needed, improving performance
and security.

```
CREATE PROCEDURE GetStudentDetails(IN student_id INT)
BEGIN
SELECT * FROM Students WHERE StudentID = student_id;
END;
```

4. TRANSACTIONS

A transaction is a group of SQL operations executed as a single unit. Either all operations succeed, or none do. This ensures data
consistency.

```
START TRANSACTION;

UPDATE Accounts SET Balance = Balance - 500 WHERE AccountID = 1;
UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 2;

COMMIT;
```

Here, money is transferred between two accounts. If any step fails, the transaction can be rolled back.

DATABASE NORMALIZATION

Normalization is the process of organizing data to reduce redundancy and improve efficiency. It divides large tables into smaller
ones and establishes relationships.

FORMS OF NORMALIZATION

* 1NF (First Normal Form): Ensure that each column holds atomic values.
* 2NF (Second Normal Form): Remove partial dependencies (every non-key attribute depends on the whole primary key).
* 3NF (Third Normal Form): Remove transitive dependencies (non-key columns depend only on primary key).
* BCNF (Boyce-Codd Normal Form): Stronger form of 3NF to eliminate anomalies.

ADVANTAGES OF SQL AND RELATIONAL DATABASES

* Structured, consistent, and reliable data storage.
* Powerful query capabilities using SQL.
* Support for ACID properties (Atomicity, Consistency, Isolation, Durability).
* Scalability and security for enterprise use.
* Standardized language used across multiple RDBMS systems.

LIMITATIONS OF SQL AND RELATIONAL DATABASES

* Not suitable for handling very large unstructured data (use NoSQL instead).
* Performance may decrease with extremely huge datasets.
* Rigid schema design—hard to modify for fast-changing requirements.
* Requires skilled database administrators for optimization.

FUTURE OF SQL DATABASES

Even with the rise of NoSQL databases, SQL remains dominant in the industry. Modern relational databases now support hybrid
models, cloud scalability, automation, and integration with AI and analytics tools.

With continuous improvements in indexing, partitioning, and cloud-native features, SQL is expected to remain the foundation of
structured data management for years to come.

CONCLUSION

SQL and relational databases form the backbone of modern software applications. From simple projects to enterprise-level systems,
they ensure data integrity, security, and performance. Whether you are a student learning databases for the first time, a
developer building projects, or a data analyst working with business insights, mastering SQL will always be a valuable skill.

SQL & RELATIONAL DATABASE INTERVIEW QUESTIONS WITH ANSWERS

Preparing for interviews requires not only theoretical knowledge but also practical understanding. Below are some frequently asked
SQL and RDBMS questions with clear answers.

1. WHAT IS SQL?

Answer: SQL (Structured Query Language) is a standard language used to manage and manipulate relational databases. It allows
creating, updating, deleting, and retrieving data.

2. DIFFERENCE BETWEEN SQL AND MYSQL?

Answer: SQL is a language, while MySQL is a database management system (RDBMS) that uses SQL as its query language.

3. WHAT ARE PRIMARY KEY AND FOREIGN KEY?

Answer: A Primary Key uniquely identifies each record in a table. A Foreign Key is a column that establishes a link between two
tables by referring to the primary key in another table.

4. WHAT ARE JOINS? NAME TYPES OF JOINS.

Answer: A JOIN is used to combine rows from two or more tables based on a related column. Types are INNER JOIN, LEFT JOIN, RIGHT
JOIN, and FULL OUTER JOIN.

5. WHAT IS NORMALIZATION?

Answer: Normalization is the process of organizing data into smaller, related tables to reduce redundancy and improve efficiency.
It involves different normal forms like 1NF, 2NF, 3NF, and BCNF.

6. WHAT IS A VIEW IN SQL?

Answer: A View is a virtual table based on a SQL query. It does not store data physically but provides a simplified interface for
complex queries.

7. WHAT ARE ACID PROPERTIES?

Answer: ACID stands for Atomicity, Consistency, Isolation, Durability. These properties ensure that database transactions are
reliable and secure.

8. DIFFERENCE BETWEEN WHERE AND HAVING?

Answer: WHERE is used to filter rows before grouping. HAVING is used to filter groups after aggregation.

9. WHAT IS THE DIFFERENCE BETWEEN DELETE, TRUNCATE, AND DROP?

* DELETE: Removes specific rows, can be rolled back.
* TRUNCATE: Removes all rows quickly, cannot be rolled back in many DBMS.
* DROP: Deletes the entire table structure and data.

10. WHAT IS THE DIFFERENCE BETWEEN CLUSTERED AND NON-CLUSTERED INDEX?

Answer: A Clustered Index sorts and stores data rows physically in the table (one per table). A Non-Clustered Index stores
pointers to data rows, allowing multiple per table.

11. WHAT IS A STORED PROCEDURE?

Answer: A Stored Procedure is a set of SQL statements stored in the database that can be executed as a single unit, improving
performance and security.

12. DIFFERENCE BETWEEN UNION AND UNION ALL?

Answer: UNION combines results of two queries and removes duplicates. UNION ALL also combines results but includes duplicates.

13. WHAT IS THE DIFFERENCE BETWEEN OLTP AND OLAP?

Answer: OLTP (Online Transaction Processing) is used for day-to-day transactions. OLAP (Online Analytical Processing) is used for
complex queries and data analysis.

14. WHAT ARE TRIGGERS?

Answer: A Trigger is a special type of stored procedure that runs automatically in response to events like INSERT, UPDATE, or
DELETE on a table.

15. EXPLAIN DIFFERENCE BETWEEN CHAR AND VARCHAR.

Answer: CHAR is fixed-length storage, meaning unused space is filled with blanks. VARCHAR is variable-length storage, saving only
required space.

16. HOW DO YOU IMPROVE DATABASE PERFORMANCE?

Answer: By using proper indexing, avoiding unnecessary joins, normalizing data, using caching, and optimizing queries with EXPLAIN
plans.

17. WHAT IS A COMPOSITE KEY?

Answer: A Composite Key is a combination of two or more columns used to uniquely identify a record.

18. DIFFERENCE BETWEEN SQL AND NOSQL?

Answer: SQL databases are structured, relational, and schema-based. NoSQL databases

are flexible, handle unstructured data, and
are good for big data and real-time applications.

19. WHAT IS A TRANSACTION IN SQL?

Answer: A Transaction is a sequence of SQL statements executed as a single unit. It ensures either all statements succeed (COMMIT)
or none (ROLLBACK).

20. EXPLAIN GROUP BY WITH EXAMPLE.

Answer: GROUP BY is used to arrange identical data into groups. Example:

```
SELECT Class, COUNT(StudentID) AS TotalStudents
FROM Students
GROUP BY Class;
```