Concept	Description	Example
Creating a Database	Creates a new database with the specified name.	CREATE DATABASE mydatabase;
Using a Database	Specifies the database to be used in subsequent queries.	USE mydatabase;
Creating a Table	Creates a new table with the specified columns and their data types.	CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(100), age INT);
Altering a Table	Adds a new column to an existing table.	ALTER TABLE employees ADD COLUMN salary DECIMAL(10,2);
Altering a Table	Drops a column from an existing table.	ALTER TABLE employees DROP COLUMN age;
SELECT	Retrieves data from a table.	SELECT * FROM employees;
WHERE	Filters rows based on a condition.	SELECT * FROM orders WHERE total_amount > 100;
INSERT INTO	Adds new rows into a table.	INSERT INTO users (name, age) VALUES ('John', 30);
UPDATE	Modifies existing rows in a table.	UPDATE customers SET city = 'New York' WHERE id = 1;
DELETE FROM	Removes rows from a table based on a condition.	DELETE FROM products WHERE stock_quantity = 0;
COUNT()	Returns the number of rows in a result set or a specific column.	SELECT COUNT(*) FROM orders;
SUM()	Calculates the sum of a numeric column.	SELECT SUM(price) FROM products;
AVG()	Computes the average of a numeric column.	SELECT AVG(rating) FROM reviews;
MIN()	Finds the minimum value in a column.	SELECT MIN(salary) FROM employees;
MAX()	Retrieves the maximum value in a column.	SELECT MAX(age) FROM users;
INNER JOIN	Retrieves rows that have matching values in both tables.	SELECT * FROM employees INNER JOIN departments ON employees.dep_id = departments.id;
LEFT JOIN (LEFT OUTER JOIN)	Returns all rows from the left table and matching rows from the right table.	SELECT * FROM customers LEFT JOIN orders ON customers.id = orders.customer_id;
RIGHT JOIN (RIGHT OUTER JOIN)	Returns all rows from the right table and matching rows from the left table.	SELECT * FROM orders RIGHT JOIN products ON orders.product_id = products.id;
Union	Combines the result sets of two or more SELECT queries.	SELECT column1 FROM table1 UNION SELECT column1 FROM table2;
Scalar Subquery	A subquery that returns a single value.	SELECT name, (SELECT MAX(age) FROM users) AS max_age FROM departments;
Correlated Subquery	A subquery that refers to a value from the outer query.	SELECT name FROM employees WHERE salary > (SELECT AVG(salary) FROM employees WHERE department = 'Sales');
Case Statement	Performs conditional logic in queries.	SELECT name, age, CASE WHEN age >= 18 THEN 'Adult' ELSE 'Minor' END AS age_group FROM users;
Aliases	Provides a temporary name to a table or column.	SELECT first_name AS 'First', last_name AS 'Last' FROM customers;
GROUP BY	Groups rows based on one or more columns.	SELECT department, COUNT(*) FROM employees GROUP BY department;
HAVING()	Filters groups based on aggregate functions.	SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 50000;
Sorting	Sorts the result set by one or more columns.	SELECT * FROM products ORDER BY price DESC;
PRIMARY KEY	Uniquely identifies a row in a table.	CREATE TABLE users (id INT PRIMARY KEY, name VARCHAR(50));
FOREIGN KEY	Establishes a link between data in two tables.	CREATE TABLE orders (order_id INT, customer_id INT, FOREIGN KEY (customer_id) REFERENCES customers(id));
UNIQUE	Ensures that all values in a column are unique.	CREATE TABLE employees (emp_id INT, email VARCHAR(50) UNIQUE);
NOT NULL	Ensures a column cannot have NULL values.	CREATE TABLE products (product_id INT, name VARCHAR(100) NOT NULL);
CHECK	Adds a condition to limit the values that can be inserted into a column.	CREATE TABLE students (student_id INT, age INT CHECK (age >= 18));
CREATE INDEX	Improves the search speed of a column.	CREATE INDEX idx_lastname ON employees (last_name);
Transactions	Allows multiple SQL statements to be executed as a single unit, ensuring data integrity and consistency.	START TRANSACTION; UPDATE account SET balance = balance - 100 WHERE account_id = 1; COMMIT;
Commit	Permanently saves the changes made during a transaction.	COMMIT;
Rollback	Reverts the changes made during a transaction to its initial state.	ROLLBACK;
Views	Virtual tables created from the result of a SELECT query.	CREATE VIEW high_salary_employees AS SELECT * FROM employees WHERE salary > 75000;
Stored Procedures	Precompiled SQL code that can be executed multiple times.	CREATE PROCEDURE sp_get_employee(IN emp_id INT) BEGIN SELECT * FROM employees WHERE emp_id = emp_id; END;
Triggers	Automatically executed SQL code in response to certain events.	CREATE TRIGGER update_stock AFTER INSERT ON orders FOR EACH ROW UPDATE products SET stock = stock - NEW.quantity WHERE product_id = NEW.product_id;
Index Optimization	Analyzing and optimizing indexes to improve query performance.	EXPLAIN SELECT * FROM products WHERE category = 'Electronics';
Subquery Optimization	Optimizing subqueries for better performance.	SELECT name FROM customers WHERE id IN (SELECT customer_id FROM orders);
Window Functions	Perform calculations across a set of rows related to the current row.	SELECT name, salary, AVG(salary) OVER (PARTITION BY department) AS avg_salary FROM employees;
Common Table Expressions (CTEs)	Temporary result sets that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement.	WITH cte AS (SELECT * FROM employees WHERE salary > 50000) SELECT * FROM cte WHERE department = 'Sales';