SQL

250+ Formulas with Examples

Basic Retrieval

- SELECT: SELECT first_name, last_name FROM employees; -- Retrieve specific columns from a table.

- SELECT *: SELECT * FROM products; -- Fetch all columns quickly when exploring data.

- DISTINCT: SELECT DISTINCT country FROM customers; -- Return unique values, removing duplicates.

- FROM (sub-query): SELECT * FROM (SELECT * FROM sales WHERE year=2025) AS y25; -- Treat a derived table as input.

- AS (alias): SELECT price AS unit_price FROM sales; -- Rename columns or tables for readability.

- Literals: SELECT 'John' AS name, 123 AS num, NULL AS nothing; -- Embed constant values directly in query.

- Expression (arith): SELECT price*qty AS total FROM order_items; -- Compute new values on the fly.

- ORDER BY: SELECT * FROM students ORDER BY score DESC; -- Sort result rows in a defined order.

- ASC / DESC: SELECT * FROM flights ORDER BY depart_dt ASC; -- Specify ascending or descending sort direction.

- LIMIT: SELECT * FROM logs LIMIT 100; -- Restrict output to first N rows for sampling.

- OFFSET: SELECT * FROM logs LIMIT 50 OFFSET 100; -- Skip rows, useful for pagination.

- FETCH FIRST: SELECT * FROM employees ORDER BY hire_date FETCH FIRST 5 ROWS ONLY; -- Standard SQL way to limit rows.

- TOP (T-SQL): SELECT TOP 10 * FROM Customers; -- SQL Server syntax to fetch first N rows.

- User Variable: SET @row_num:=0; SELECT @row_num:=@row_num+1 AS n, name FROM people; -- Store and reuse values within session.

- USE (DB): USE sakila; -- Switch current database context. - SHOW DATABASES: SHOW DATABASES; -- List databases present on the server.

- SHOW TABLES: SHOW TABLES FROM sakila; -- List tables in a database. - DESCRIBE / DESC: DESC employees; -- Display table schema quickly.

- EXPLAIN: EXPLAIN SELECT * FROM sales WHERE id=1; -- Inspect execution plan for performance tuning. - HELP: HELP SELECT; -- Get built-in MySQL documentation for a keyword.

- COMMENT in DDL: CREATE TABLE t(id INT COMMENT 'Primary key'); -- Attach human-readable notes to schema objects.

- Inline Comments: -- Single line comment -- Document code or disable lines during testing.

- FORCE INDEX(): SELECT * FROM t FORCE INDEX(idx_col) WHERE col=1; -- Override optimizer to use a specific index.

- SQL_MODE: SET sql_mode='STRICT_ALL_TABLES'; -- Change SQL behaviour (e.g., strictness) for the session.

- SELECT INTO OUTFILE: SELECT * FROM employees INTO OUTFILE '/tmp/emp.csv' FIELDS TERMINATED BY ','; -- Export query result to an external file.

Filtering & Boolean Logic

- WHERE: SELECT * FROM orders WHERE amount > 500; -- Filter rows based on a condition.
- AND / OR / NOT: SELECT * FROM users WHERE active=1 AND NOT country='IN'; -- Combine multiple filter conditions logically.
- Comparison Operators: SELECT * FROM items WHERE qty <> 0; -- Compare column values to constants or other columns.
- BETWEEN: WHERE order_date BETWEEN '2025-01-01' AND '2025-01-31'; -- Select values inside an inclusive range.
- NOT BETWEEN: WHERE age NOT BETWEEN 18 AND 30; -- Exclude values that fall within a range.
- IN: WHERE status IN ('NEW','PENDING'); -- Match against a set of literal values.
- NOT IN: WHERE status NOT IN ('CANCELLED'); -- Exclude rows with values in a given set.
- LIKE: WHERE name LIKE 'A%'; -- Search text using wildcard patterns.
- NOT LIKE: WHERE code NOT LIKE 'ERR%'; -- Filter out rows matching a pattern.
- REGEXP / RLIKE: WHERE email REGEXP '@gmail\.com$'; -- Match text with regular expressions.
- NOT REGEXP: WHERE phone NOT REGEXP '^\+91'; -- Exclude rows matching a regex pattern.
- IS NULL: WHERE manager_id IS NULL; -- Find rows with NULL (missing) values.
- IS NOT NULL: WHERE manager_id IS NOT NULL; -- Find rows where value exists (not NULL).
- EXISTS: WHERE EXISTS (SELECT 1 FROM payments p WHERE p.order_id=o.id); -- Test for existence of related rows in a subquery.
- NOT EXISTS: WHERE NOT EXISTS (SELECT 1 FROM blacklist b WHERE b.user_id=u.id); -- Ensure no related rows exist.
- ANY / SOME / ALL: WHERE salary > ALL (SELECT salary FROM staff WHERE dept='HR'); -- Compare a value against a subquery result set.
- CASE WHEN: SELECT CASE WHEN score>=60 THEN 'Pass' ELSE 'Fail' END AS result FROM tests; -- Create conditional computed columns.
- IF() (MySQL): SELECT IF(qty>0,'In Stock','Out') FROM products; -- MySQL-specific inline conditional expression.
- COALESCE: SELECT COALESCE(nickname,name) AS display_name FROM users; -- Return first non-NULL argument.
- NULLIF: SELECT NULLIF(col1,col2) AS diff FROM t; -- Return NULL if two expressions are equal.
- GREATEST / LEAST: SELECT GREATEST(q1,q2,q3) AS best FROM scores; -- Pick the largest or smallest value from a list.
- INTERVAL(): SELECT INTERVAL(score,50,60,70) AS grade_band FROM exams; -- Return index of range into which value falls.
- Row Sub-query: WHERE (a,b) IN (SELECT x,y FROM t2); -- Compare multiple columns as a composite key.
- WITH RECURSIVE: WITH RECURSIVE nums AS (...) SELECT * FROM nums; -- Generate sequences or hierarchical data via CTE.

Set & Join Operations

- INNER JOIN: SELECT e.name,d.name FROM emp e INNER JOIN dept d ON e.dept_id=d.id; -- Combine rows that satisfy join condition in both tables.
- LEFT JOIN: SELECT * FROM a LEFT JOIN b ON a.id=b.id; -- Keep all rows from left table, matching from right when present.
- RIGHT JOIN: SELECT * FROM a RIGHT JOIN b ON a.id=b.id; -- Keep all rows from right table, matching from left when present.
- CROSS JOIN: SELECT * FROM colors CROSS JOIN sizes; -- Produce Cartesian product of two tables.
- NATURAL JOIN: SELECT * FROM t1 NATURAL JOIN t2; -- Automatically join tables on columns with same names.
- SELF JOIN: SELECT a.name,b.name FROM employees a JOIN employees b ON a.manager_id=b.id; -- Join a table to itself to compare rows.
- USING(): SELECT * FROM orders JOIN customers USING(customer_id); -- Shorthand join when column names are identical.
- UNION: SELECT city FROM a UNION SELECT city FROM b; -- Vertically concatenate result sets removing duplicates.
- UNION ALL: SELECT city FROM a UNION ALL SELECT city FROM b; -- Combine result sets keeping duplicates for performance.
- INTERSECT: SELECT id FROM a INTERSECT SELECT id FROM b; -- Return rows present in both queries.
- EXCEPT: SELECT id FROM a EXCEPT SELECT id FROM b; -- Return rows from first query absent in second.
- APPLY (T-SQL): SELECT * FROM a CROSS APPLY dbo.fn(a.id); -- Invoke table-valued function for each row.
- Join ON TRUE: SELECT * FROM a LEFT JOIN b ON TRUE; -- Emulate FULL OUTER JOIN in MySQL via union of left and right joins.
- WINDOW clause: SELECT sum(v) OVER w FROM t WINDOW w AS (PARTITION BY c); -- Define reusable window specifications.
- LATERAL: SELECT * FROM t1, LATERAL (SELECT MAX(val) m FROM t2 WHERE t2.id=t1.id) AS x; -- Allow subquery to reference preceding table.
- MERGE: MERGE INTO dest USING src ON(dest.id=src.id) WHEN MATCHED THEN UPDATE SET ...; -- Perform conditional insert/update (upsert) in ANSI SQL or supported engines.
- PIVOT / UNPIVOT: SELECT ... PIVOT(SUM(amt) FOR qtr IN ('Q1','Q2','Q3','Q4')) AS p; -- Rotate rows to columns or vice versa for reporting.

Aggregation & Grouping

- GROUP BY: SELECT dept,COUNT(*) c FROM emp GROUP BY dept; -- Aggregate rows sharing same attribute.
- HAVING: SELECT dept,COUNT(*) c FROM emp GROUP BY dept HAVING c>5; -- Filter groups based on aggregate conditions.
- COUNT(): SELECT COUNT(*) FROM t; -- Return number of rows or non-NULL values.
- SUM() / AVG() / MIN() / MAX(): SELECT AVG(salary) FROM emp; -- Compute total, average, minimum, or maximum of numeric columns.
- GROUP_CONCAT(): SELECT GROUP_CONCAT(name ORDER BY name) FROM products; -- Concatenate strings from grouped rows.
- ROLLUP: GROUP BY year,quarter WITH ROLLUP; -- Generate subtotals along hierarchy levels.
- CUBE: GROUP BY CUBE(year,quarter); -- Produce subtotals for all combinations of grouping columns.
- GROUPING SETS: GROUP BY GROUPING SETS ((year,quarter),(year),()); -- Specify custom combinations for aggregation.
- ANY_VALUE(): SELECT year,ANY_VALUE(product) FROM sales GROUP BY year; -- Return arbitrary non-aggregate column when only_full_group_by is on.
- STDDEV() / VARIANCE(): SELECT STDDEV(population) FROM cities; -- Compute statistical dispersion measures.

Window Functions

- ROW_NUMBER(): SELECT ROW_NUMBER() OVER(ORDER BY score DESC) AS rnk FROM exams; -- Assign unique sequential number to each row in window order.
- RANK(): SELECT RANK() OVER(PARTITION BY dept ORDER BY salary DESC) r FROM emp; -- Assign rank with gaps for ties within partition.
- DENSE_RANK(): SELECT DENSE_RANK() OVER(PARTITION BY dept ORDER BY salary DESC) dr FROM emp; -- Assign consecutive ranks without gaps.
- NTILE(): SELECT NTILE(4) OVER(ORDER BY salary) quartile FROM emp; -- Distribute rows into equal buckets.
- LAG(): SELECT year,sales,LAG(sales) OVER(ORDER BY year) prev FROM rev; -- Access value from preceding row.
- LEAD(): SELECT year,sales,LEAD(sales) OVER(ORDER BY year) next FROM rev; -- Access value from following row.
- FIRST_VALUE(): SELECT FIRST_VALUE(price) OVER(ORDER BY dt) AS first_price FROM btc; -- Return first value in window frame.
- LAST_VALUE(): SELECT LAST_VALUE(price) OVER(ORDER BY dt RANGE BETWEEN UNBOUNDED FOLLOWING AND UNBOUNDED FOLLOWING) last_price FROM btc; -- Return last value in frame.
- NTH_VALUE(): SELECT NTH_VALUE(sales,3) OVER(ORDER BY year) AS third_year FROM rev; -- Return nth value in frame.
- CUME_DIST(): SELECT CUME_DIST() OVER(ORDER BY score) cd FROM exams; -- Calculate cumulative distribution.
- PERCENT_RANK(): SELECT PERCENT_RANK() OVER(ORDER BY score) pr FROM exams; -- Compute relative rank of row.
- ROWS BETWEEN: SUM(sales) OVER(PARTITION BY prod ORDER BY dt ROWS BETWEEN 3 PRECEDING AND CURRENT ROW); -- Define moving window frame for aggregation.
- RANGE BETWEEN: AVG(price) OVER(ORDER BY dt RANGE BETWEEN INTERVAL 7 DAY PRECEDING AND CURRENT ROW);
 -- Use value-based bounds for window.
- EXCLUDE CURRENT ROW: AVG(score) OVER(ORDER BY score DESC EXCLUDE CURRENT ROW); -- Remove current row from frame calculation.
- WINDOW NAMED clause: SELECT SUM(v) OVER w FROM t WINDOW w AS (PARTITION BY c); -- Reuse window specification across functions.

String Functions

- CONCAT(): SELECT CONCAT(first,' ',last) AS fullname FROM users; -- Combine two or more strings together.
- CONCAT_WS(): SELECT CONCAT_WS('-',year,month,day) FROM dates; -- Concatenate with a separator automatically.
- SUBSTRING(): SELECT SUBSTRING(code,1,3) FROM sku; -- Extract substring from a position.
- LEFT(): SELECT LEFT(name,5) FROM customers; -- Get leftmost characters.
- RIGHT(): SELECT RIGHT(name,4) FROM customers; -- Get rightmost characters.
- LENGTH(): SELECT LENGTH(title) FROM books; -- Return byte length of string.
- CHAR_LENGTH(): SELECT CHAR_LENGTH(title) FROM books; -- Return number of characters.
- OCTET_LENGTH(): SELECT OCTET_LENGTH(title) FROM books; -- Return length in bytes per multibyte char.
- TRIM(): SELECT TRIM(' xyz ') AS t; -- Remove leading and trailing spaces.
- LTRIM(): SELECT LTRIM(' abc'); -- Remove leading spaces.
- RTRIM(): SELECT RTRIM('abc '); -- Remove trailing spaces.
- LOWER(): SELECT LOWER(city) FROM locations; -- Convert text to lowercase.
- UPPER(): SELECT UPPER(city) FROM locations; -- Convert text to uppercase.
- REPLACE(): SELECT REPLACE(txt,'foo','bar') FROM msgs; -- Replace occurrences of substring.
- INSERT(): SELECT INSERT('ABCDE',3,2,'--') AS new; -- Replace part of string starting at position.
- SUBSTRING_INDEX(): SELECT SUBSTRING_INDEX(email,'@',1) FROM users; -- Return substring before or after delimiter.
- LOCATE(): SELECT LOCATE('ab',text) FROM docs; -- Find position of substring.
- INSTR(): SELECT INSTR(text,'ab') FROM docs; -- Same as LOCATE but arg order reversed.
- POSITION(): SELECT POSITION('ab' IN text) FROM docs; -- ANSI alternative to LOCATE.
- REPEAT(): SELECT REPEAT('*',5); -- Repeat a string N times.
- REVERSE(): SELECT REVERSE(name) FROM people; -- Return string with characters reversed.
- LPAD(): SELECT LPAD(num,4,'0') FROM seq; -- Pad string on left to target length.
- RPAD(): SELECT RPAD(num,4,'0') FROM seq; -- Pad on right.
- FORMAT(): SELECT FORMAT(12345.678,2); -- Format number with locale commas.
- QUOTE(): SELECT QUOTE("O'Reilly"); -- Escape string for SQL.
- ELT(): SELECT ELT(2,'x','y','z'); -- Return Nth element of list.
- FIELD(): SELECT FIELD('y','x','y','z'); -- Return position of value in list.

String Functions

- MAKE_SET(): SELECT MAKE_SET(5,'a','b','c','d'); -- Return set of strings for bits set.
- FIND_IN_SET(): SELECT FIND_IN_SET('b','a,b,c'); -- Locate string in comma-separated list.
- SOUNDEX(): SELECT SOUNDEX('juice'); -- Return phonetic code for string.
- DIFFERENCE(): SELECT DIFFERENCE('juice','juce'); -- Compare Soundex codes (SQL Server).
- HEX(): SELECT HEX('abc'); -- Convert string to hexadecimal.
- UNHEX(): SELECT UNHEX('616263'); -- Convert hex to binary string.
- UUID(): SELECT UUID(); -- Generate a universally unique identifier.
- SPACE(): SELECT CONCAT('Hi',SPACE(3),'Bye'); -- Insert N spaces.
- CHAR(): SELECT CHAR(65,66,67); -- Return characters from ASCII codes.
- CHARSET(): SELECT CHARSET('abc'); -- Show string's character set.
- COLLATION(): SELECT COLLATION('abc'); -- Show collation used for comparison.
- MD5(): SELECT MD5('password'); -- Generate 128-bit hash of string.
- SHA2(): SELECT SHA2('password',256); -- Generate SHA-2 hash of specified length.

Numeric Functions

- ABS(): SELECT ABS(-9); -- Return absolute (positive) value.
- SIGN(): SELECT SIGN(-9); -- Return sign of number (-1,0,1).
- CEIL(): SELECT CEIL(4.2); -- Round up to nearest integer.
- FLOOR(): SELECT FLOOR(4.8); -- Round down to nearest integer.
- ROUND(): SELECT ROUND(4.567,2); -- Round to given decimal places.
- TRUNCATE(): SELECT TRUNCATE(123.456,1); -- Truncate without rounding.
- MOD(): SELECT MOD(11,4); -- Return remainder of division.
- % operator: SELECT 11 % 4; -- Alternate syntax for MOD.
- POWER(): SELECT POWER(2,8); -- Raise number to a power.
- EXP(): SELECT EXP(1); -- Return e raised to a power.
- SQRT(): SELECT SQRT(25); -- Return square root.
- LOG(): SELECT LOG(10,100); -- Logarithm with custom base.
- LN(): SELECT LN(7); -- Natural logarithm.
- LOG10(): SELECT LOG10(100); -- Base-10 logarithm.
- RAND(): SELECT RAND(); -- Generate pseudo random number.
- PI(): SELECT PI(); -- Return constant .
- SIN(): SELECT SIN(PI()/2); -- Trigonometric sine.
- COS(): SELECT COS(0); -- Cosine of angle.
- TAN(): SELECT TAN(PI()/4); -- Tangent of angle.
- ACOS(): SELECT ACOS(1); -- Inverse cosine.
- DEGREES(): SELECT DEGREES(PI()); -- Convert radians to degrees.
- RADIANS(): SELECT RADIANS(180); -- Convert degrees to radians.
- LEAST(): SELECT LEAST(5,9,3); -- Return smallest argument.
- GREATEST(): SELECT GREATEST(5,9,3); -- Return largest argument.
- CONV(): SELECT CONV(15,10,2); -- Convert number between bases.
- BIT_COUNT(): SELECT BIT_COUNT(7); -- Count bits set to 1.
- CRC32(): SELECT CRC32('abc'); -- Compute CRC32 checksum.
- WIDTH_BUCKET(): SELECT WIDTH_BUCKET(score,0,100,4) FROM exams; -- Assign numeric bucket category.

Date & Time Functions

- NOW(): SELECT NOW(); -- Current date and time in session timezone.

- CURRENT_TIMESTAMP: SELECT CURRENT_TIMESTAMP; -- ANSI synonym for NOW().

- CURDATE(): SELECT CURDATE(); -- Current date (no time).

- CURTIME(): SELECT CURTIME(); -- Current time (no date).

- SYSDATE(): SELECT SYSDATE(); -- Time when function executes, not start of query.

- UTC_TIMESTAMP(): SELECT UTC_TIMESTAMP(); -- Current UTC date and time.

- DATE(): SELECT DATE('2025-04-23 10:20:00'); -- Extract date part from datetime.

- TIME(): SELECT TIME('2025-04-23 10:20:00'); -- Extract time part.

- YEAR(): SELECT YEAR(order_date) FROM orders; -- Extract year part.

- MONTH(): SELECT MONTH(order_date) FROM orders; -- Extract month number.

- DAY(): SELECT DAY(order_date) FROM orders; -- Extract day of month.

- DAYNAME(): SELECT DAYNAME(order_date) FROM orders; -- Return weekday name.

- MONTHNAME(): SELECT MONTHNAME(order_date) FROM orders; -- Return month name.

- DAYOFWEEK(): SELECT DAYOFWEEK(NOW()); -- Return weekday index (1-7).

- DAYOFYEAR(): SELECT DAYOFYEAR(NOW()); -- Return day number in year.

- WEEK(): SELECT WEEK(NOW()); -- ISO week number.

- QUARTER(): SELECT QUARTER(NOW()); -- Quarter of year.

- HOUR(): SELECT HOUR(NOW()); -- Hour part.

- MINUTE(): SELECT MINUTE(NOW()); -- Minute part.

- SECOND(): SELECT SECOND(NOW()); -- Second part.

- TIMESTAMP(): SELECT TIMESTAMP('2025-01-01','12:34:56'); -- Create datetime from date and time.

- TIMESTAMPDIFF(): SELECT TIMESTAMPDIFF(DAY,'2025-01-01','2025-01-10'); -- Difference between two timestamps.

- TIMESTAMPADD(): SELECT TIMESTAMPADD(MONTH,3,NOW()); -- Add interval to timestamp.

- DATEDIFF(): SELECT DATEDIFF('2025-02-01','2025-01-01'); -- Difference in days.

- STR_TO_DATE(): SELECT STR_TO_DATE('23-04-2025','%d-%m-%Y'); -- Convert string to date via format.

- DATE_FORMAT(): SELECT DATE_FORMAT(NOW(),'%d-%b-%Y'); -- Format date/time to string.

- ADDDATE(): SELECT ADDDATE('2025-01-01', INTERVAL 7 DAY); -- Add interval to date.

- SUBDATE(): SELECT SUBDATE('2025-01-08',INTERVAL 7 DAY); -- Subtract interval.

- DATE_ADD(): SELECT DATE_ADD(NOW(), INTERVAL 1 HOUR); -- Alias of ADDDATE for datetime.

- DATE_SUB(): SELECT DATE_SUB(NOW(), INTERVAL 1 HOUR); -- Alias of SUBDATE for datetime.

- MAKEDATE(): SELECT MAKEDATE(2025,60); -- Create date from year and day of year.

Date & Time Functions

- MAKETIME(): SELECT MAKETIME(12,30,0); -- Create time value.

- EXTRACT(): SELECT EXTRACT(ISOYEAR FROM NOW()); -- Pull specified part from date/time.

- LAST_DAY(): SELECT LAST_DAY('2025-02-10'); -- Return last day of month.

- CONVERT_TZ(): SELECT CONVERT_TZ(NOW(),'UTC','Asia/Kolkata'); -- Convert datetime between zones.

- FROM_DAYS(): SELECT FROM_DAYS(737791); -- Convert day number to date.

- TO_DAYS(): SELECT TO_DAYS(NOW()); -- Convert date to number of days since zero date.

- UNIX_TIMESTAMP(): SELECT UNIX_TIMESTAMP(NOW()); -- Get seconds since Unix epoch.

- FROM_UNIXTIME(): SELECT FROM_UNIXTIME(1700000000); -- Convert epoch seconds to datetime.

- GET_FORMAT(): SELECT DATE_FORMAT(NOW(), GET_FORMAT(DATE,'EUR')); -- Return format string for locale.

Control Flow Statements

- CASE (searched): SELECT CASE WHEN score>=80 THEN 'A' WHEN score>=60 THEN 'B' END AS grade FROM exams; -- Return values conditionally in SELECT.
- IF(): SELECT IF(active,'Y','N') FROM users; -- MySQL-specific ternary conditional.
- IFNULL(): SELECT IFNULL(phone,'N/A') FROM contacts; -- Substitute NULL with alternate value.
- COALESCE(): SELECT COALESCE(mobile,landline,'N/A') FROM contacts; -- Return first non-NULL among arguments.
- RETURN: RETURN total; -- Exit stored function and return value.
- SIGNAL: SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT='Oops'; -- Raise custom error inside stored routine.
- RESIGNAL: RESIGNAL; -- Pass on caught error with possible changes.
- HANDLER: DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1; -- Define action when specific condition occurs in stored routine.
- DECLARE ...: DECLARE v INT DEFAULT 0; -- Declare variable or cursor inside routine.
- LEAVE / ITERATE: LEAVE my_loop; -- Control flow inside loops in routines.

Data Manipulation (DML)

- INSERT: INSERT INTO emp(name,salary) VALUES('Ram',50000); -- Add new row(s) to a table.

- INSERT SET: INSERT INTO emp SET name='Asha',salary=40000; -- MySQL variant offering column=value syntax.

- ON DUPLICATE KEY UPDATE: INSERT INTO stats(id,val) VALUES(1,10) ON DUPLICATE KEY UPDATE val=VALUES(val); -- Upsert values when unique key conflict occurs.

- INSERT IGNORE: INSERT IGNORE INTO users(id,email) VALUES(1,'x@y.com'); -- Skip insert errors such as duplicates.

- REPLACE INTO: REPLACE INTO cache(key,val) VALUES('a','b'); -- Delete conflicting row then insert new data.

- UPDATE: UPDATE emp SET salary=salary*1.05 WHERE dept='IT'; -- Modify existing rows.

- UPDATE JOIN: UPDATE emp e JOIN dept d ON e.dept_id=d.id SET e.dept_name=d.name; -- Update based on data from another table.

- DELETE: DELETE FROM logs WHERE created<DATE_SUB(NOW(),INTERVAL 30 DAY); -- Remove rows matching condition.

- DELETE JOIN: DELETE a FROM a JOIN b ON a.id=b.id WHERE b.flag=0; -- Delete with join criteria across tables.

- TRUNCATE: TRUNCATE TABLE temp_data; -- Remove all rows quickly and reset auto-increment.

- LOAD DATA INFILE: LOAD DATA INFILE '/tmp/file.csv' INTO TABLE t FIELDS TERMINATED BY ','; -- Bulk-load data from file.

- CALL (proc): CALL raise_salary(1001,5000); -- Execute stored procedure.

Data Definition (DDL)

- CREATE DATABASE: CREATE DATABASE salesdb CHARACTER SET utf8mb4; -- Create new database with chosen charset.
- ALTER DATABASE: ALTER DATABASE salesdb COLLATE utf8mb4_0900_ai_ci; -- Change database properties.
- DROP DATABASE: DROP DATABASE olddb; -- Remove database and all contents.
- CREATE TABLE: CREATE TABLE emp(id INT PRIMARY KEY,name VARCHAR(50)); -- Define structure to store data.
- CREATE TEMPORARY TABLE: CREATE TEMPORARY TABLE tmp SELECT * FROM emp LIMIT 0; -- Create table that exists only for session.
- ALTER TABLE ADD: ALTER TABLE emp ADD COLUMN joined DATE; -- Add new column to existing table.
- ALTER TABLE MODIFY: ALTER TABLE emp MODIFY COLUMN name VARCHAR(100); -- Change column definition.
- ALTER TABLE DROP: ALTER TABLE emp DROP COLUMN obsolete; -- Remove column.
- RENAME TABLE: RENAME TABLE temp TO archive_emp; -- Change table name.
- DROP TABLE: DROP TABLE IF EXISTS junk; -- Delete table permanently.
- CREATE VIEW: CREATE VIEW v_active AS SELECT * FROM users WHERE active=1; -- Save query as virtual table.
- ALTER VIEW: ALTER VIEW v_active AS SELECT id,name FROM users WHERE active=1; -- Change view definition.
- DROP VIEW: DROP VIEW v_active; -- Remove view.
- CREATE INDEX: CREATE INDEX idx_dept ON emp(dept); -- Speed up queries filtering by column.
- CREATE UNIQUE INDEX: CREATE UNIQUE INDEX uidx_email ON users(email); -- Enforce uniqueness and speed lookup.
- DROP INDEX: DROP INDEX idx_dept ON emp; -- Remove index.
- FULLTEXT INDEX: CREATE FULLTEXT INDEX f_idx ON docs(content); -- Enable full-text search on text columns.
- SPATIAL INDEX: CREATE SPATIAL INDEX s_idx ON geom(geo); -- Accelerate spatial queries on geometry.
- CREATE SEQUENCE: CREATE SEQUENCE seq START WITH 1 INCREMENT BY 1; -- Generate sequential numbers (MariaDB/Postgres).
- AUTO_INCREMENT: id INT AUTO_INCREMENT PRIMARY KEY; -- Automatically assign incrementing IDs.

Constraints & Relationships

- PRIMARY KEY: PRIMARY KEY(id) -- Uniquely identify each row in table.

- UNIQUE: UNIQUE(email) -- Enforce unique values in column.

- NOT NULL: name VARCHAR(100) NOT NULL -- Require column to always have a value.

- DEFAULT: status CHAR(1) DEFAULT 'A' -- Set value when none provided.

- CHECK: CHECK(qty>=0) -- Validate column values against rule.

- FOREIGN KEY: FOREIGN KEY(dept_id) REFERENCES dept(id) -- Maintain referential integrity between tables.

- ON DELETE CASCADE: ON DELETE CASCADE -- Automatically delete child rows when parent removed.

- ON UPDATE SET NULL: ON UPDATE SET NULL -- Set foreign key to NULL when parent key changes.

Transactions & Locks

- SET autocommit: SET autocommit=0; -- Turn off auto-committing for manual control.

- START TRANSACTION: START TRANSACTION; -- Begin explicit transaction block.

- SAVEPOINT: SAVEPOINT sp1; -- Mark point for partial rollback.

- ROLLBACK TO: ROLLBACK TO sp1; -- Undo to savepoint without ending transaction.

- COMMIT: COMMIT; -- Persist all changes in current transaction.

- ROLLBACK: ROLLBACK; -- Undo all changes in transaction.

- LOCK TABLES: LOCK TABLES emp WRITE; -- Prevent concurrent modification.

- UNLOCK TABLES: UNLOCK TABLES; -- Release explicit table locks.

- SET TRANSACTION ISOLATION LEVEL: SET TRANSACTION ISOLATION LEVEL READ COMMITTED; -- Control concurrency phenomena visibility.

- SHOW ENGINE INNODB STATUS: SHOW ENGINE INNODB STATUS; -- Inspect InnoDB lock and wait information.

Stored Routines & Scheduler

- DELIMITER: DELIMITER $$ ... $$ DELIMITER ; -- Change statement terminator to define routines.

- CREATE PROCEDURE: CREATE PROCEDURE p() BEGIN SELECT NOW(); END; -- Encapsulate reusable SQL logic.

- ALTER PROCEDURE: ALTER PROCEDURE p COMMENT 'Returns DT'; -- Modify procedure metadata.

- DROP PROCEDURE: DROP PROCEDURE p; -- Remove stored procedure.

- CREATE FUNCTION: CREATE FUNCTION f(x INT) RETURNS INT RETURN x*2; -- Create user-defined scalar function.

- DROP FUNCTION: DROP FUNCTION f; -- Remove function.

- CREATE TRIGGER: CREATE TRIGGER trg BEFORE INSERT ON emp FOR EACH ROW SET NEW.created=NOW(); -- Execute logic automatically on data change.

- DROP TRIGGER: DROP TRIGGER trg; -- Remove trigger.

- CREATE EVENT: CREATE EVENT e_cleanup ON SCHEDULE EVERY 1 DAY DO DELETE FROM logs WHERE created<DATE_SUB(NOW(),INTERVAL 30 DAY); -- Schedule recurring job.

- ALTER EVENT DISABLE: ALTER EVENT e_cleanup DISABLE; -- Enable or disable scheduled event.

- DROP EVENT: DROP EVENT e_cleanup; -- Delete scheduled event.

- PREPARE / EXECUTE / DEALLOCATE: PREPARE stmt FROM 'SELECT ?+?'; EXECUTE stmt USING @a,@b; DEALLOCATE PREPARE stmt; -- Run dynamic SQL safely.

JSON Functions

- JSON_OBJECT(): SELECT JSON_OBJECT('id',id,'name',name) FROM users; -- Build JSON object from key-value pairs.
- JSON_ARRAY(): SELECT JSON_ARRAY(col1,col2) FROM t; -- Create JSON array from values.
- JSON_EXTRACT(): SELECT JSON_EXTRACT(json,'$.path') FROM t; -- Retrieve value from JSON document.
- -> operator: SELECT json->'$.path' FROM t; -- Shorthand for JSON_EXTRACT in MySQL.
- JSON_UNQUOTE(): SELECT JSON_UNQUOTE('"text"'); -- Remove quotes from JSON string.
- JSON_SET(): SELECT JSON_SET(json,'$.a',1) FROM t; -- Insert or replace value at path.
- JSON_INSERT(): SELECT JSON_INSERT(json,'$.a',1) FROM t; -- Add value only if path doesn't exist.
- JSON_REPLACE(): SELECT JSON_REPLACE(json,'$.a',1) FROM t; -- Replace only if path exists.
- JSON_REMOVE(): SELECT JSON_REMOVE(json,'$.a') FROM t; -- Delete value at path.
- JSON_MERGE_PATCH(): SELECT JSON_MERGE_PATCH(j1,j2); -- Merge JSON documents (RFC 7396).
- JSON_CONTAINS(): SELECT JSON_CONTAINS(json, '1', '$.a'); -- Test if JSON contains value.
- JSON_CONTAINS_PATH(): SELECT JSON_CONTAINS_PATH(json,'one','$.a'); -- Check if path(s) exist.
- JSON_KEYS(): SELECT JSON_KEYS(json,'$.a') FROM t; -- Return array of keys.
- JSON_LENGTH(): SELECT JSON_LENGTH(json,'$.array') FROM t; -- Count elements.
- JSON_TYPE(): SELECT JSON_TYPE(json,'$.a') FROM t; -- Return data type at path.
- JSON_VALID(): SELECT JSON_VALID(json) FROM t; -- Check if string is valid JSON.
- JSON_TABLE(): SELECT * FROM JSON_TABLE(json,'$[*]' COLUMNS(id INT PATH '$.id')) AS jt; -- Transform JSON array/object to relational rows.
- JSON_PRETTY(): SELECT JSON_PRETTY(json); -- Pretty-print JSON for readability.

Admin & Security

- CREATE USER: CREATE USER 'u'@'%' IDENTIFIED BY 'p'; -- Add new database login account.

- ALTER USER: ALTER USER 'u'@'%' IDENTIFIED WITH 'mysql_native_password' BY 'np'; -- Change authentication method or password.

- DROP USER: DROP USER 'u'@'%'; -- Remove user account.

- GRANT: GRANT SELECT,INSERT ON db.* TO 'u'@'%'; -- Assign privileges.

- REVOKE: REVOKE INSERT ON db.* FROM 'u'@'%'; -- Remove privileges.

- SHOW GRANTS: SHOW GRANTS FOR 'u'@'%'; -- Display privilege statements for user.

- FLUSH PRIVILEGES: FLUSH PRIVILEGES; -- Reload privilege tables after manual edits.

- SET PASSWORD: SET PASSWORD = PASSWORD('new'); -- Change current user password.

- RESET MASTER: RESET MASTER; -- Clear binary logs in replication master.

- KILL: KILL 1234; -- Terminate running thread.

- SHOW PROCESSLIST: SHOW PROCESSLIST; -- View active connections and queries.

- SHOW STATUS: SHOW STATUS LIKE 'Threads%'; -- Check server status counters.

- SHOW VARIABLES: SHOW VARIABLES LIKE 'max_connections'; -- Inspect configuration parameters.

- SHOW ENGINE INNODB STATUS: SHOW ENGINE INNODB STATUS; -- Detailed diagnostics for InnoDB.

- PING: SELECT 1; -- Test connection; server responds OK.