SQL QUICK REFERENCE

SELECT

```
SELECT * FROM users;
SELECT name, email FROM users;
SELECT DISTINCT city FROM users;
SELECT name AS full_name FROM users;
```

WHERE

Comparison Operators

`= <> (!=)`	Equal / not equal
`< > <= >=`	Comparison operators
`AND OR NOT`	Logical operators
`IS NULL / IS NOT NULL`	Null checks

Pattern Matching

```
SELECT * FROM users WHERE name LIKE 'A%';
-- % = any chars, _ = single char
SELECT * FROM users WHERE age IN (20, 25, 30);
SELECT * FROM users WHERE age BETWEEN 18 AND 30;
```

JOIN

Join Types

`INNER JOIN`	Rows matching in both tables
`LEFT JOIN`	All left rows + matching right
`RIGHT JOIN`	All right rows + matching left
`FULL OUTER JOIN`	All rows from both tables
`CROSS JOIN`	Cartesian product of both tables

Join Syntax

```
SELECT u.name, o.total
FROM users u
INNER JOIN orders o ON u.id = o.user_id;

SELECT u.name, o.total
FROM users u
LEFT JOIN orders o ON u.id = o.user_id;
```

INSERT / UPDATE / DELETE

Insert

```
INSERT INTO users (name, email)
VALUES ('Alice', 'alice@example.com');

INSERT INTO users (name, email) VALUES
('Bob', 'bob@ex.com'),
('Carol', 'carol@ex.com');
```

Update

```
UPDATE users SET email = 'new@ex.com'
WHERE id = 1;
```

Delete

```
DELETE FROM users WHERE id = 1;
DELETE FROM users; -- delete all rows
```

CREATE TABLE

Syntax

```
CREATE TABLE users (
  id    INTEGER PRIMARY KEY AUTOINCREMENT,
  name  TEXT NOT NULL,
  email TEXT UNIQUE,
  age   INTEGER DEFAULT 0,
  score REAL
);
```

Common Data Types

`INTEGER`	Whole numbers
`REAL`	Floating-point numbers
`TEXT`	String / text data
`BLOB`	Binary data
`BOOLEAN`	TRUE / FALSE (stored as 0/1)
`DATE / DATETIME`	Date and timestamp values

Constraints

`PRIMARY KEY`	Unique row identifier
`NOT NULL`	Value required
`UNIQUE`	No duplicate values
`DEFAULT val`	Default value if omitted
`CHECK (expr)`	Custom validation rule
`FOREIGN KEY`	Reference to another table

Aggregate Functions

`COUNT(*)`	Number of rows
`COUNT(col)`	Non-null values in column
`SUM(col)`	Sum of numeric column
`AVG(col)`	Average of numeric column
`MIN(col)`	Minimum value
`MAX(col)`	Maximum value

Example

```
SELECT COUNT(*) AS total,
       AVG(age) AS avg_age,
       MAX(score) AS top_score
FROM users;
```

GROUP BY / HAVING

```
SELECT city, COUNT(*) AS num_users
FROM users
GROUP BY city;

SELECT city, AVG(age) AS avg_age
FROM users
GROUP BY city
HAVING AVG(age) > 25;
```

WHERE filters rows before grouping; HAVING filters groups after aggregation

ORDER BY / LIMIT

```
SELECT * FROM users ORDER BY name ASC;
SELECT * FROM users ORDER BY age DESC;
SELECT * FROM users
ORDER BY city, name
LIMIT 10 OFFSET 20;  -- skip 20, take 10
```

Subqueries

In WHERE Clause

```
SELECT name FROM users
WHERE id IN (
  SELECT user_id FROM orders
  WHERE total > 100
);
```

As Derived Table

```
SELECT city, avg_age FROM (
  SELECT city, AVG(age) AS avg_age
  FROM users GROUP BY city
) WHERE avg_age > 30;
```

Indexes

```
CREATE INDEX idx_name ON users(name);
CREATE UNIQUE INDEX idx_email
  ON users(email);
DROP INDEX idx_name;
```

When to Index

`Columns in WHERE`	Speed up filtering
`Columns in JOIN ON`	Speed up join lookups
`Columns in ORDER BY`	Speed up sorting
`High-cardinality cols`	Many unique values benefit most