

# SQL QUICK REFERENCE

SELECT, JOIN, subqueries, indexes, transactions

## SELECT

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

## WHERE

### Comparison Operators

<b>= &lt;&gt; (!=)</b>	Equal / not equal
<b>&lt; &gt; &lt;= &gt;=</b>	Comparison operators
<b>AND OR NOT</b>	Logical operators
<b>IS NULL / IS NOT NULL</b>	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

<b>INNER JOIN</b>	Rows matching in both tables
<b>LEFT JOIN</b>	All left rows + matching right
<b>RIGHT JOIN</b>	All right rows + matching left
<b>FULL OUTER JOIN</b>	All rows from both tables
<b>CROSS JOIN</b>	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

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

### Constraints

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

## Aggregate Functions

<b>COUNT(*)</b>	Number of rows
<b>COUNT(col)</b>	Non-null values in column
<b>SUM(col)</b>	Sum of numeric column
<b>AVG(col)</b>	Average of numeric column
<b>MIN(col)</b>	Minimum value
<b>MAX(col)</b>	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

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