CockroachDB SQL Cheatsheet

A quick reference guide to CockroachDB's SQL dialect, covering data definition, manipulation, querying, and transaction management. Useful for both beginners and experienced users looking to quickly recall syntax and best practices.

Data Definition Language (DDL)

Creating Databases

Syntax: `CREATE DATABASE <database_name>;`

Creates a new database.

Example:

```
CREATE DATABASE my_app;
```

Creating a database if it does not exist: `CREATE DATABASE IF NOT EXISTS <database_name>;`

Creates a new database only if one doesn't already exist.

Example:

```
CREATE DATABASE IF NOT EXISTS my_app;
```

Creating Tables

Syntax: `CREATE TABLE <table_name> (<column_definitions>);`

Creates a new table.

Example:

```
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  username STRING NOT NULL,
  email STRING UNIQUE NOT NULL,
  created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT now()
);
```

Data Types: Common data types include `INT`, `STRING`, `BOOL`, `DECIMAL`, `DATE`, `TIMESTAMP`, `UUID`, `JSONB`.

Constraints: `PRIMARY KEY`, `NOT NULL`, `UNIQUE`, `DEFAULT`, `CHECK`.

Creating a table with foreign key:

```
CREATE TABLE orders (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users (id) ON DELETE CASCADE,
  order_date TIMESTAMP WITHOUT TIME ZONE DEFAULT now()
);
```

Altering Tables

Syntax: `ALTER TABLE <table_name> ADD COLUMN <column_name> <data_type> [constraints];`

Adds a new column to an existing table.

Example:

```
ALTER TABLE users ADD COLUMN age INT;
```

Syntax: `ALTER TABLE <table_name> DROP COLUMN <column_name>;`

Removes a column from an existing table.

Example:

```
ALTER TABLE users DROP COLUMN age;
```

Syntax: `ALTER TABLE <table_name> RENAME COLUMN <old_name> TO <new_name>;`

Renames a column in an existing table.

Example:

```
ALTER TABLE users RENAME COLUMN username TO user_name;
```

Data Manipulation Language (DML)

Inserting Data

Syntax: `INSERT INTO <table_name> (<column_names>) VALUES (<values>);`

Inserts new rows into a table.

Example:

```
INSERT INTO users (username, email) VALUES ('john_doe', 'john.doe@example.com');
```

Inserting multiple rows:

```
INSERT INTO users (username, email) VALUES ('jane_doe', 'jane.doe@example.com'), ('jim_smith', 'jim.smith@example.com');
```

Inserting all columns: If inserting into all columns, you can omit the column names.

```
INSERT INTO users VALUES (gen_random_uuid(), 'new_user', 'new.user@example.com', now());
```

Updating Data

Syntax: `UPDATE <table_name> SET <column_name> = <value> WHERE <condition>;`

Updates existing rows in a table.

Example:

```
UPDATE users SET username = 'john.updated' WHERE email = 'john.doe@example.com';
```

Updating multiple columns:

```
UPDATE users SET username = 'john.updated', email = 'john.updated@example.com' WHERE id = '...';
```

Deleting Data

Syntax: `DELETE FROM <table_name> WHERE <condition>;`

Deletes rows from a table.

Example:

```
DELETE FROM users WHERE email = 'john.doe@example.com';
```

Deleting all rows (truncate): Use with caution!

```
DELETE FROM users;
```

Data Querying (SELECT)

Basic SELECT Statements

Syntax: `SELECT <column_names> FROM <table_name> WHERE <condition>;`

Retrieves data from a table.

Example:

```
SELECT id, username, email FROM users WHERE username LIKE 'j%';
```

Selecting all columns:

```
SELECT * FROM users;
```

Using aliases:

```
SELECT username AS user_name, email AS user_email FROM users;
```

Filtering and Sorting

WHERE Clause:

```
SELECT * FROM users WHERE age > 18 AND city = 'New York';
```

ORDER BY Clause:

```
SELECT * FROM users ORDER BY username ASC, created_at DESC;
```

LIMIT and OFFSET:

```
SELECT * FROM users ORDER BY created_at DESC LIMIT 10 OFFSET 20;
```

Aggregate Functions

Common functions: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`.

Example:

```
SELECT COUNT(*) FROM users WHERE age > 18;
```

GROUP BY Clause:

```
SELECT city, AVG(age) FROM users GROUP BY city;
```

HAVING Clause: Filters groups based on a condition.

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

Transactions

Transaction Management

Starting a transaction: `BEGIN;`

Marks the beginning of a transaction block.

Example:

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;
```

Committing a transaction: `COMMIT;`

Saves all changes made during the transaction.

Rolling back a transaction: `ROLLBACK;`

Discards all changes made during the transaction.

Savepoints:

```
SAVEPOINT my_savepoint;
-- Perform operations
ROLLBACK TO SAVEPOINT my_savepoint;
RELEASE SAVEPOINT my_savepoint;
```

Isolation Levels

CockroachDB supports `SERIALIZABLE` isolation, which is the strongest level.

It prevents phenomena like dirty reads, non-repeatable reads, and phantom reads.

Concurrency Control

CockroachDB uses optimistic concurrency control, which assumes that multiple transactions can frequently complete without interfering with each other.