SQL

CHEAT SHEET

SWIPE→

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query Data In Columns C1, C2 From A Table

```
SELECT * FROM t;
```
Query All Rows And Columns From A Table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query Data And Filter Rows With A Condition

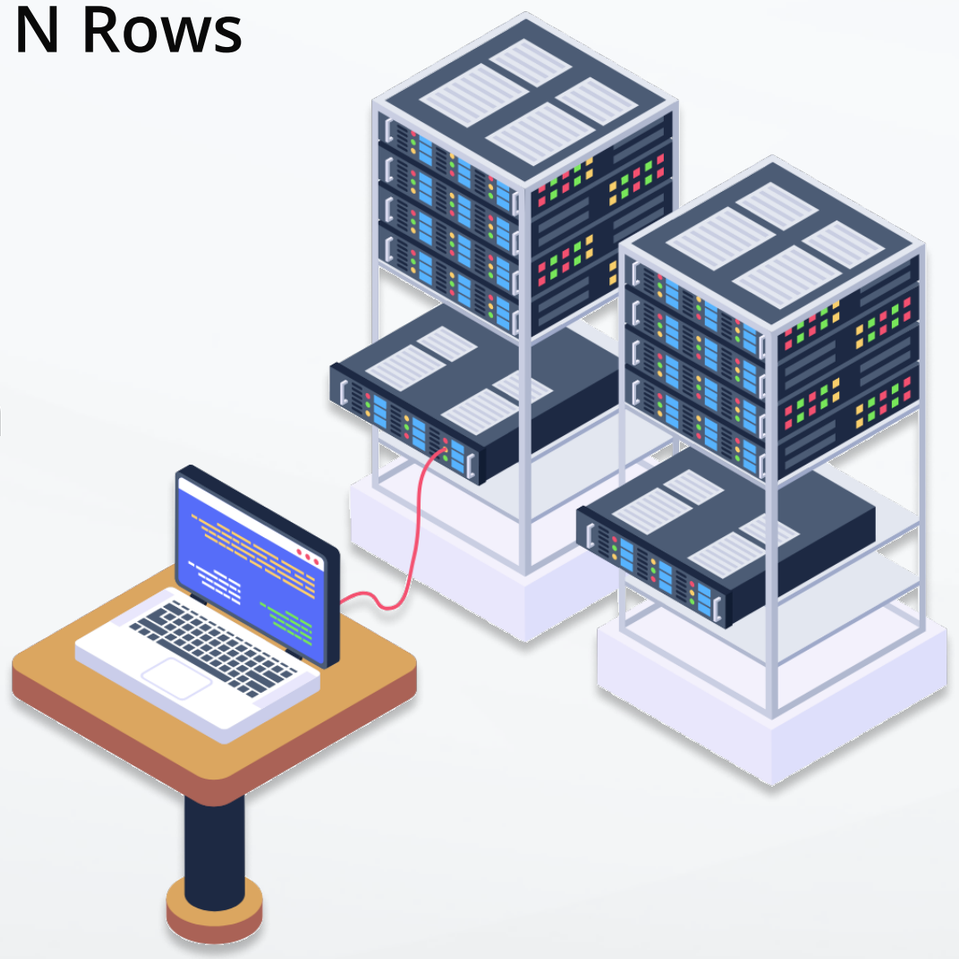
```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query Distinct Rows From A Table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort The Result Set In Ascending Or Descending Order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip Offset Of Rows And Return The Next N Rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group Rows Using An Aggregate Function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter Groups Using HAVING Clause



QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```

Inner Join T1 And T2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```

Left Join T1 And T1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```

Right Join T1 And T2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```

Perform Full Outer Join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```

Produce A Cartesian Product Of Rows In Tables

```
SELECT c1, c2
FROM t1, t2;
```

Another Way To Perform Cross Join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```

Join T1 To Itself Using INNER JOIN Clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```

Combine Rows From Two Queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT C1, C2 FROM T2;
```

Return The Intersection Of Two Queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```

Subtract A Result Set From Another Result Set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```

Query Rows Using Pattern Matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```

Query Rows In A List

```
SELECT c1, c2 FROM t
WHERE c1 BETWEEN low AND high;
```

Query Rows Between Two Values

```
SELECT c1, c2 FROM t
WHERE c1 IS [NOT] NULL;
```

Check If Values In A Table Is NULL Or Not

MANAGING TABLES

```
CREATE TABLE t (
     id INT PRIMARY KEY,
     name VARCHAR NOT NULL,
     price INT DEFAULT 0
);
```

Create a new table with three columns

```
DROP TABLE t ;
```

Delete the table from the database

```
ALTER TABLE t ADD column;
```

Add a new column to the table

```
ALTER TABLE t DROP COLUMN c ;
```

Drop column c from the table

```
ALTER TABLE t ADD constraint;
```

Add a constraint

```
ALTER TABLE t DROP constraint;
```

Drop a constraint

```
ALTER TABLE t1 RENAME TO t2;
```

Rename a table from t1 to t2

```
ALTER TABLE t1 RENAME c1 TO c2 ;
```

Rename column c1 to c2

```
TRUNCATE TABLE t;
```

Remove all data in a table

USING SQL CONSTRAINTS

```
CREATE TABLE t(
     c1 INT, c2 INT, c3 VARCHAR,
     PRIMARY KEY (c1,c2)
);
```

Set c1 and c2 as a primary key

```
CREATE TABLE t1(
c1 INT PRIMARY KEY,
c2 INT,
FOREIGN KEY (c2) REFERENCES t2(c2)
);
```

Set c2 column as a foreign key

```
CREATE TABLE t(
c1 INT, c1 INT,
UNIQUE(c2,c3)
);
```

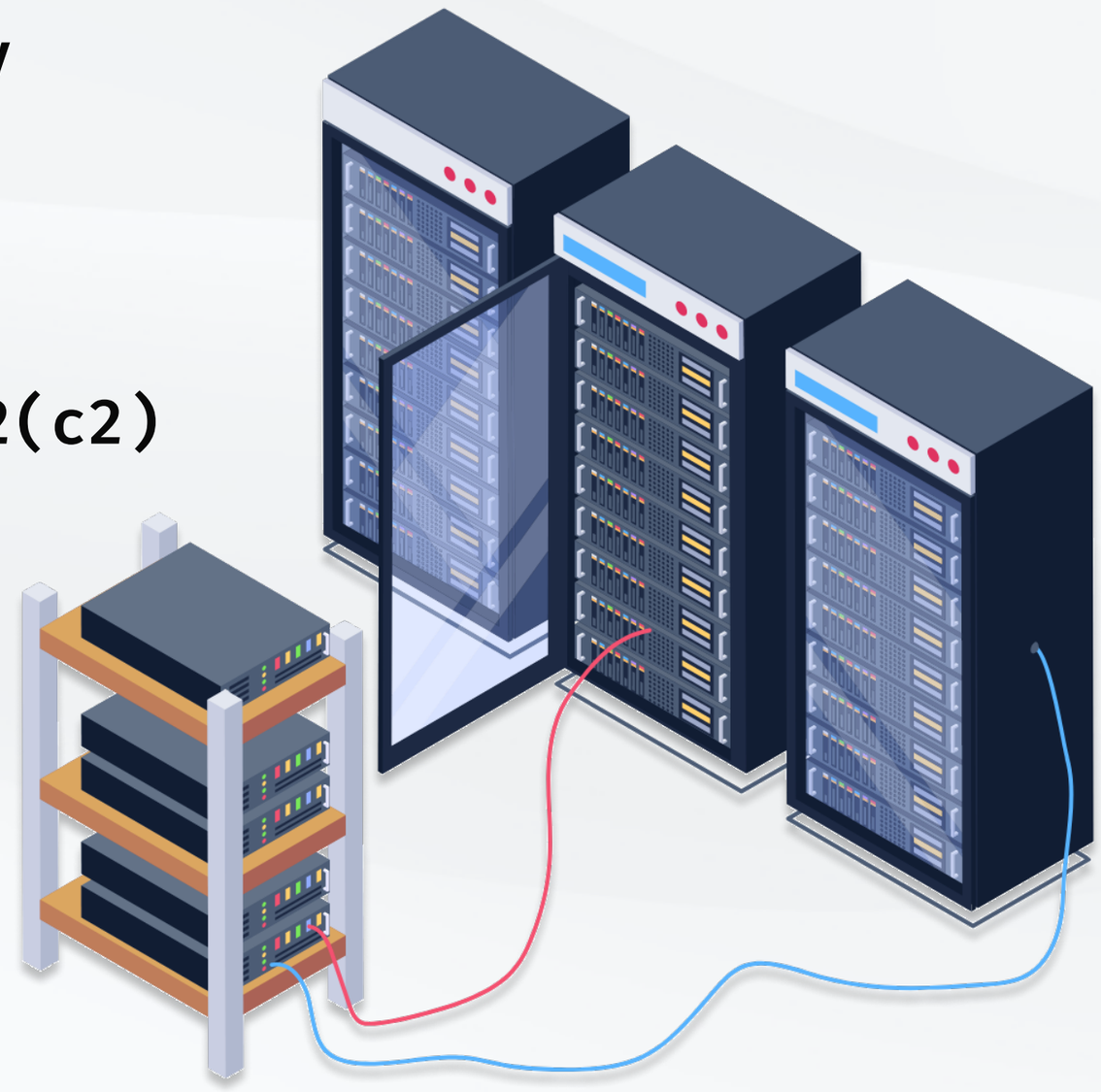
Make the values in c1 and c2 unique

```
CREATE TABLE t(
c1 INT, c2 INT,
CHECK(c1> 0 AND c1 ⩾ c2)
);
```

Ensure c1 > 0 and values in c1 >= c2

```
CREATE TABLE t(
c1 INT PRIMARY KEY,
c2 VARCHAR NOT NULL
);
```

Set values in c2 column not NULL

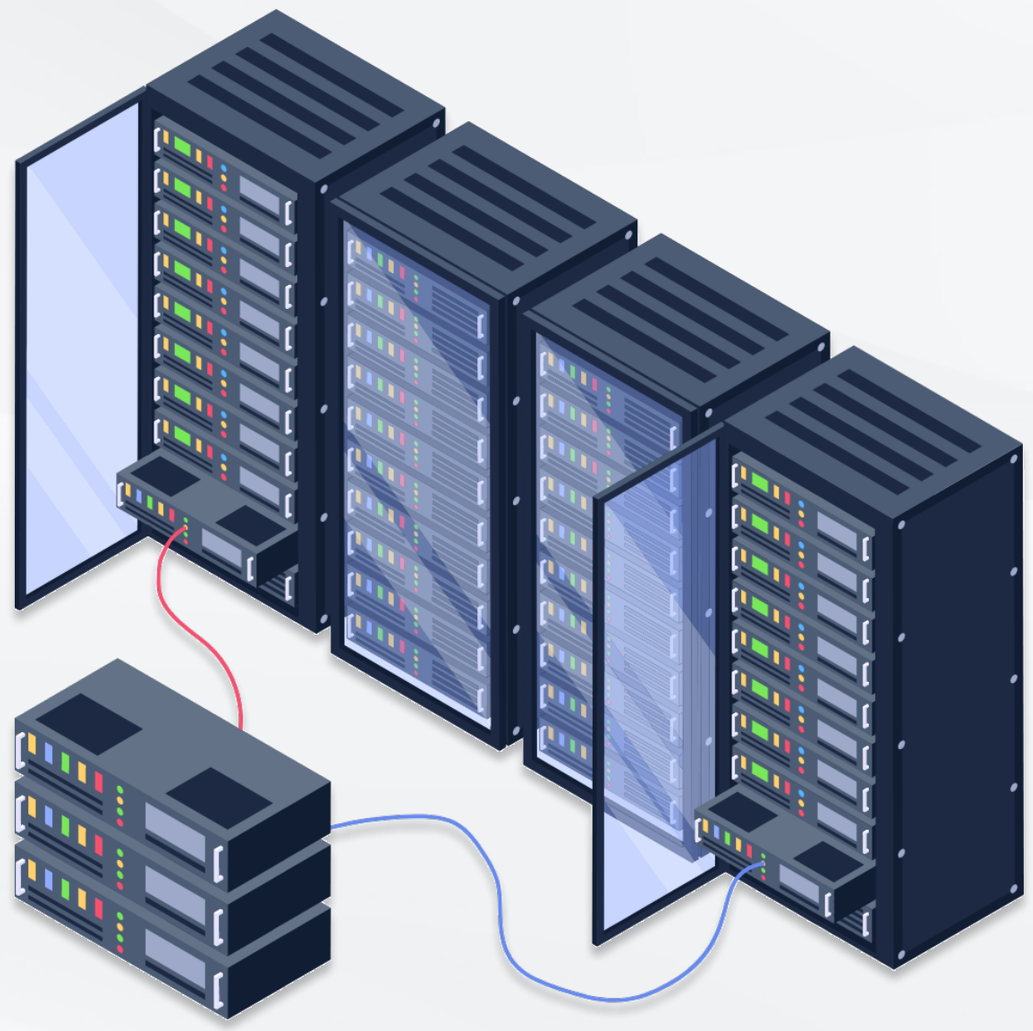


MODIFYING DATA

```
INSERT INTO t(column_list)
VALUES(value_list);
```
Insert one row into a table

```
INSERT INTO t(column_list)
VALUES (value_list),
       (value_list), ....;
```
Insert multiple rows into a table

```
INSERT INTO t1(column_list)
SELECT column_list
FROM t2;
```
Insert rows from t2 into t1



```
UPDATE t
SET c1 = new_value;
```
Update new value in the column c1 for all rows

```
UPDATE t
SET c1 = new_value,
    c2 = new_value
WHERE condition;
```
Update values in the column c1, c2 that match the condition

```
DELETE FROM t;
```
Delete all data in a table

```
DELETE FROM t
WHERE condition;
```
Delete subset of rows in a table

MANAGING VIEWS

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
```
Create a new view that consists of c1 and c2

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
WITH [CASCADED | LOCAL] CHECK OPTION;
```
Create a new view with check option

```
CREATE RECURSIVE VIEW v
AS
select-statement -- anchor part
UNION [ALL]
select-statement; -- recursive part
```
Create a recursive view

```
CREATE TEMPORARY VIEW v
AS
SELECT c1, c2
FROM t;
```
Create a temporary view

```
DROP VIEW view_name
```
Delete a view

MANAGING TRIGGERS

```
CREATE OR MODIFY TRIGGER trigger_name
WHEN EVENT
ON table_name TRIGGER_TYPE
EXECUTE stored_procedure;
```

Create or modify a trigger

WHEN

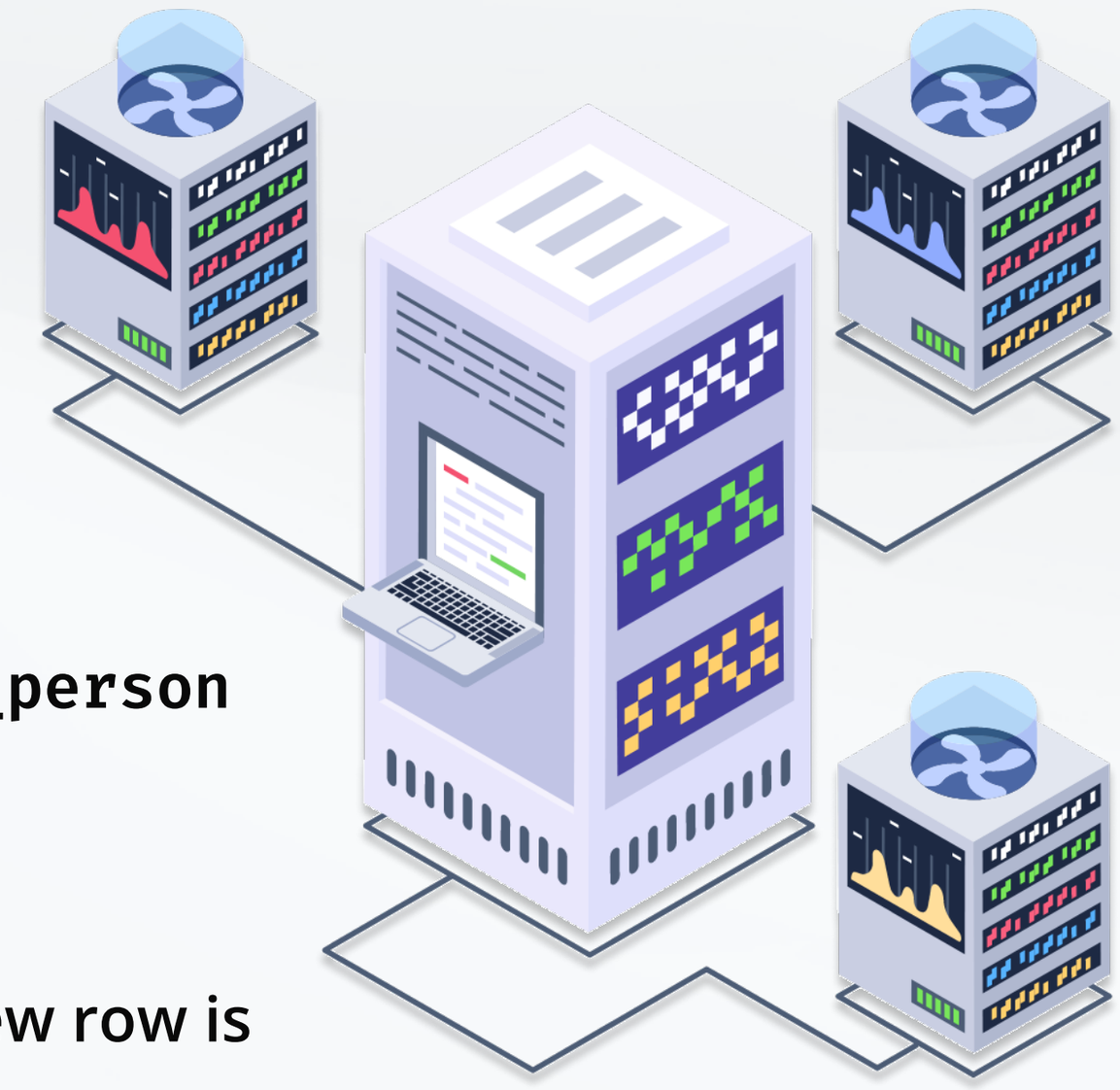
- **BEFORE** – invoke before the event occurs
- **AFTER** – invoke after the event occurs

EVENT

- **INSERT** – invoke for INSERT
- **UPDATE** – invoke for UPDATE
- **DELETE** – invoke for DELETE

TRIGGER_TYPE

- FOR EACH ROW
- FOR EACH STATEMENT



```
CREATE TRIGGER before_insert_person
BEFORE INSERT
ON person FOR EACH ROW
EXECUTE stored_procedure;
```

Create a trigger invoked before a new row is inserted into the person table

```
DROP TRIGGER trigger_name
```

Delete a specific trigger

SKILL<S/ASH>

DID YOU LIKE THIS POST?

TELL US IN THE COMMENTS BELOW!

DROP A FOLLOW FOR MORE SUCH VALUABLE CONTENT!