Warmup:

START AT 3:07

UPDATE SO

I have two tables, one called pokemon and one called skills. Each table has a “ID” field, and pokemon has a “name,” “level,” and”type” field. For the pokemon table, what are the most appropriate data types?

ID – integer, primary key, autoincrement Name – text / varchar (n) Level – integer

type – varchar / text

If skills contains an “ID” and a “skill_name” field, create a SQL query that returns all Pokemon names and their skill names.

1	Pikachu	Lightning
2	Mudkip	Water

```
SELECT pokemon.name, skills.skill_name
FROM pokemon JOIN skills ON
  pokemon.ID = skills.ID
```

Return the total number of Pokemon of type “ghost.”

```
SELECT COUNT(*) FROM pokemon WHERE type = "ghost"
```

Return all Pokemon names similar to “mud.”

```
SELECT name FROM pokemon WHERE name LIKE %mud%
```

Return all Pokemon with level > 5.

```
SELECT name FROM pokemon WHERE level > 5
```

CS50 Section 7

In order to build increasingly complex websites, we depend on a **database** to store information long-term. The simplest form of a database with which we are all likely familiar is a basic spreadsheet, organized into rows and columns, tabs (tables), and individual files (databases).

SQL is a programming language whose purpose is to *query* databases (perform operations on them).

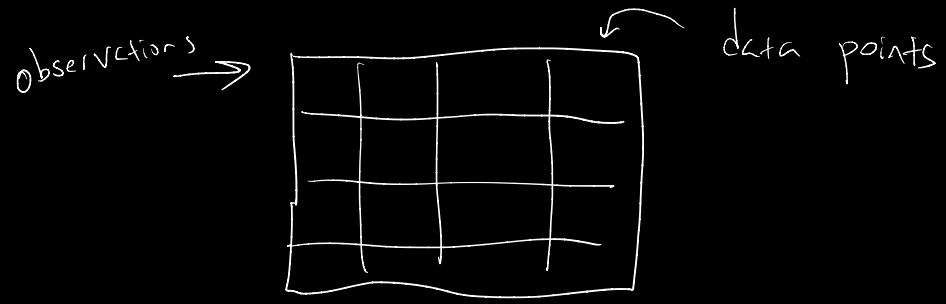
After you create a database, you create one or more **tables**.

For each table, you specify all of the **columns** in the table.

When new information is added to the database, the new information (typically) goes into a new **row**.

There are many data types that can be stored in a SQL database. This is just a small sample.

INT	SMALLINT	TINYINT	MEDIUMINT	BIGINT
DECIMAL	FLOAT	BIT	DATE	TIME
DATETIME	TIMESTAMP	CHAR	VARCHAR	BINARY
BLOB	TEXT	ENUM	GEOMETRY	LINESTRING



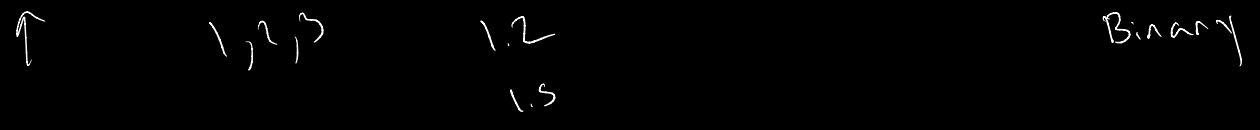
After you create a database, you create one or more **tables**.

For each table, you specify all of the **columns** in the table.

When new information is added to the database, the new information (typically) goes into a new **row**.

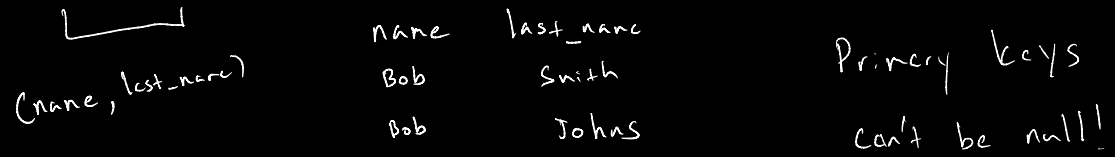
In SQLite, which we'll use in this course, we can consolidate these various datatypes into a few more general classes (though underlying types still exist)

NULL	INTEGER	REAL	TEXT	BLOB



Another consideration is choosing a column to be a **primary key**, guaranteed to be unique across rows. A good primary key makes subsequent table operations much easier.

You can also have a *joint primary key*, a combination of two or more columns where the combination is guaranteed to be unique.



SQL is a programming language, but it has a limited vocabulary that we'll use.

Another consideration is choosing a column to be a **primary key**, guaranteed to be unique across rows. A good primary key makes subsequent table operations much easier.

You can also have a *joint primary key*, a combination of two or more columns where the combination is guaranteed to be unique.

SQL is a programming language, but it has a limited vocabulary that we'll use.

INSERT

SELECT

UPDATE

DELETE

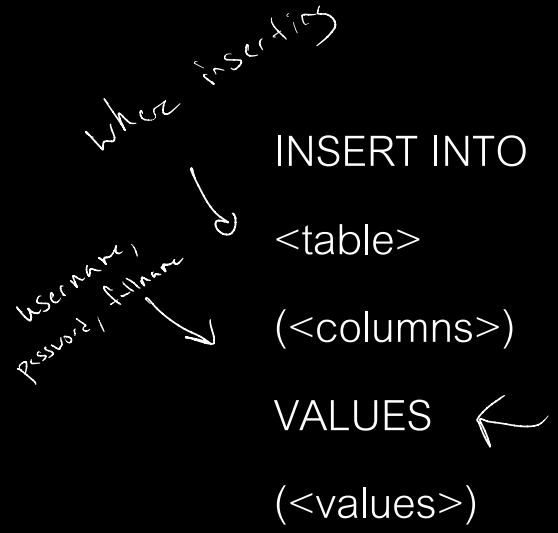
users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza

An INSERT query adds information to a table.




```
INSERT INTO noms
(mother, username) VALUES
("luca", "mom")
```

An INSERT query adds information to a table.

```
INSERT INTO
users
(username, password, fullname)
VALUES
('newman', 'USMAIL', 'Newman')
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza

When defining the column that ultimately is your primary key, it's usually a good idea for that column to be an integer.

Moreover, you can configure that column to **autoincrement**, so it will pre-populate that column for you automatically when rows are added, eliminating the risk that you'll accidentally try to insert something with a duplicate value.

A SELECT query extracts information from a table.

SELECT

<columns>

FROM

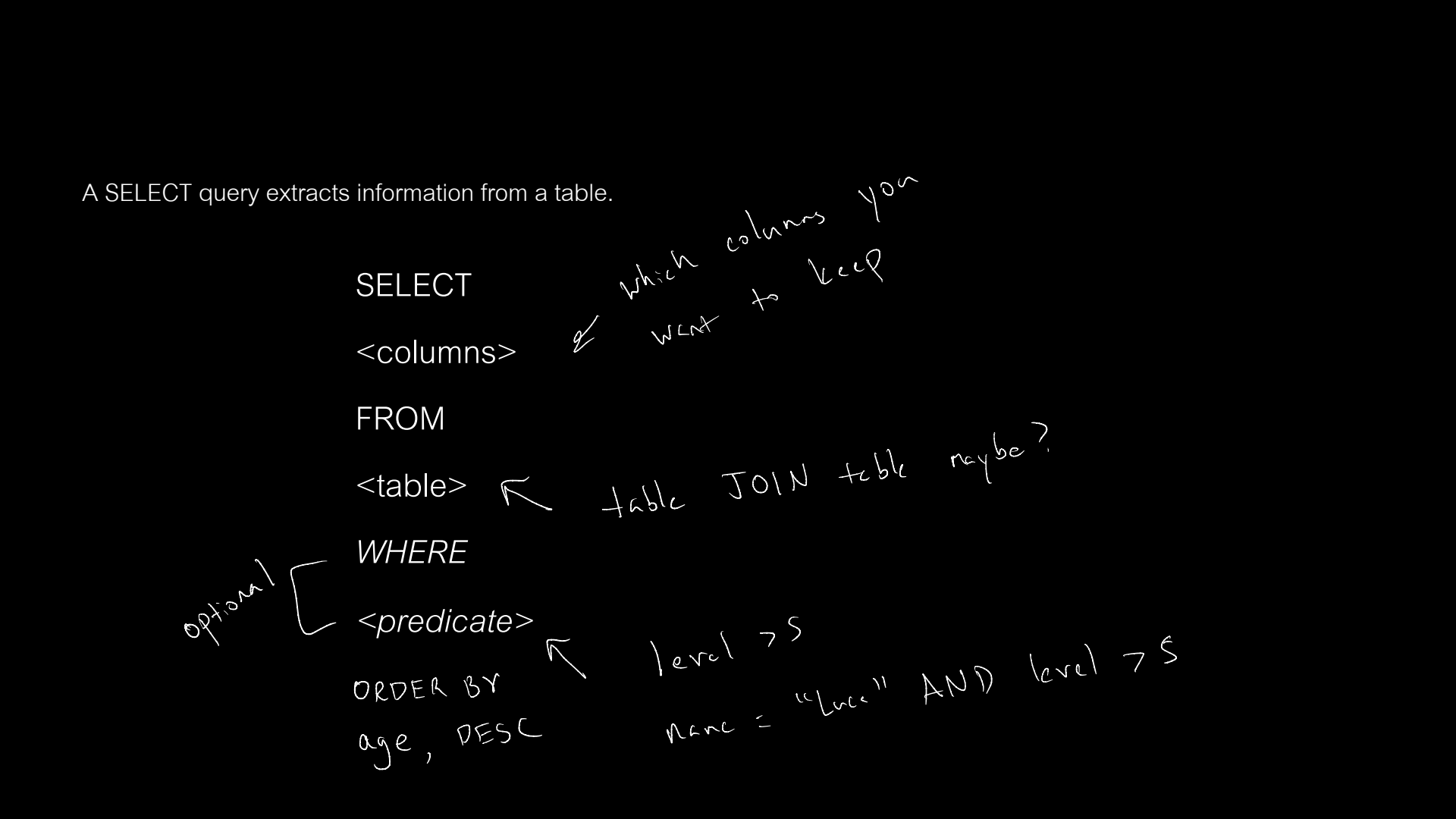
<table>

WHERE

<predicate>

ORDER BY

age, DESC



A SELECT query extracts information from a table.

```
SELECT
idnum, fullname
FROM
users
```

users

idnum	username	password	fullname
10	jerry	fus!!!!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

Databases empower us to organize information into tables efficiently.

We don’t always need to store every possible relevant piece of information in the same table, but rather we can use relationships across tables to connect all the pieces of data we need.

Let’s imagine we need to get a user’s full name (from the *users* table) and their mother’s name (from the *moms* table).

A SELECT (JOIN) query extracts information from multiple tables.

table1.column_name
table2.column_name

```
SELECT
<columns>
FROM
<table1>
JOIN
<table2>
ON
<predicate>
WHERE
```

One table linked on a specific column

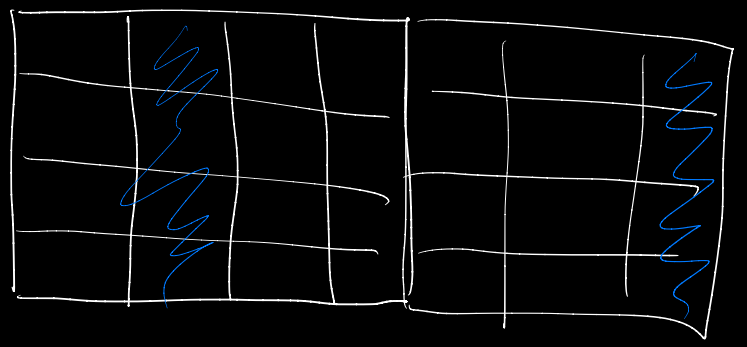
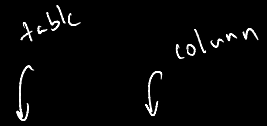


Table 1 Table 2

A SELECT (JOIN) query extracts information from multiple tables.




```
SELECT
users.fullname, moms.mother
FROM
users
JOIN
moms
ON
users.username = moms.username
```

- A SELECT (JOIN) query extracts information from multiple tables.

```
SELECT
users.fullname, moms.mother
FROM
users
JOIN
moms
ON
users.username = moms.username
```

users

idnum	username	password	fullname
10	jerry	fus!!!!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

users & moms

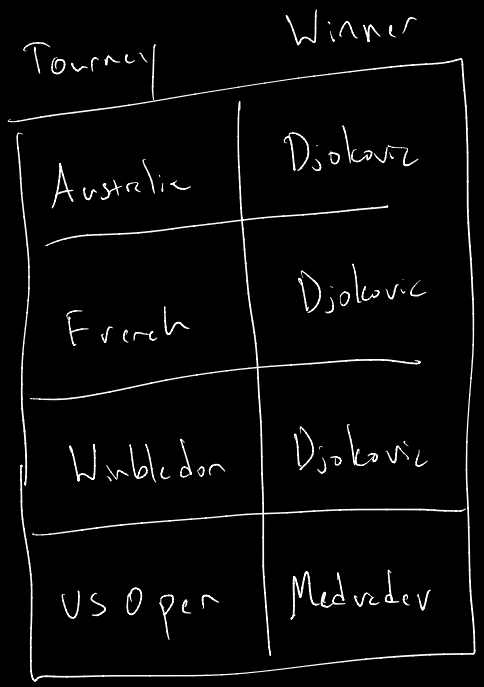
users.idnum	users.username moms.username	users.password	users.fullname	moms.mother
10	jerry	fus!!!!	Jerry Seinfeld	Helen Seinfeld
11	gcostanza	b0sc0	George Costanza	Estelle Costanza

SELECT users.fullname, moms.mother FROM joined table

- An UPDATE query modifies information in a table.

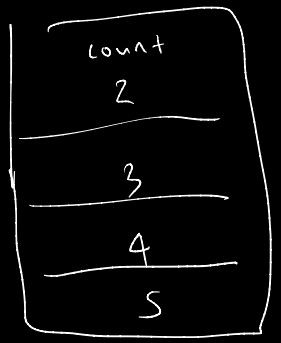
```
UPDATE
  tennis
SET
Winner = "Medvedev"
  WHERE
Tourney = "US Open"
```

```
UPDATE
<table>
SET
<column> = <value>
WHERE
<predicate>
```



An UPDATE query modifies information in a table.

```
UPDATE
users
SET
password = 'yadayada'
WHERE
idnum = 10
```



count = count + 1

users

idnum	username	password	fullname
10	jerry	yadayada	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

A DELETE query removes information from a table.

DELETE FROM

<table>

WHERE

<predicate>

DELETE FROM

pokemon

WHERE

name = "Pikachu"

A DELETE query removes information from a table.

```
DELETE FROM
users
WHERE
username = 'newman'
```

users

idnum	username	password	fullname
10	jerry	fus!ll!	Jerry Seinfeld
11	gcostanza	b0sc0	George Costanza
12	newman	USMAIL	Newman

moms

username	mother
jerry	Helen Seinfeld
gcostanza	Estelle Costanza
kramer	Babs Kramer

SELECT * FROM users WHERE username = ? AND password = ?", username, password

Regroup at 4:06

1 Person Likes:

1 cur_likes = rows[0]["likes"] 5 5

2 update likes = cur_likes + 1 6

SQL Injection Attacks and Race Conditions

2nd Person Likes:

3 cur_likes = rows[0]["likes"] 6

4 update likes = cur_likes + 1 7 ?

```
rows = db.execute(f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'")
```

Table of Users

lucadwong

lucadwong'--

password

"SELECT * FROM users WHERE username = 'lucadwong'--' AND password = '{password}'"

BEGIN TRANSACTION

Queries

COMMIT

Python and SQL!

```
from CS50 import SQL

rows = db.execute("SELECT * FROM users WHERE username = ? AND password = ?", username, password)
```

Hints

In your folder, run *sqlite fiftyville.db* to start running queries!

Running *.schema table_name* will show you the fieldnames and types!

Running *.tables* will list all of the tables in the fiftyville database!

LAB

Database of Songs:

AVG( )

JOIN ←

LIMIT N

└┘

only returns the first N results

```
SELECT name FROM songs;

SELECT name FROM SONGS ORDER BY
tempo;

SELECT name FROM songs ORDER BY
duration_ms DESC LIMIT 5;

SELECT name FROM songs WHERE
danceability > 0.75 AND energy > 0.75 AND
valence > 0.75
```

AVG

LAB

```
SELECT AVG(energy) FROM songs;
```

Nested Selects

```
SELECT name FROM songs
WHERE artist_id =
(
    SELECT id
    FROM artists
    WHERE name = "Post Malone"
);
```

Joins

```
SELECT name FROM
songs JOIN artists ON
songs.artist_id = artists.artist_id
WHERE artists.name =
"Post Malone"
```

LAB

```
SELECT AVG(energy) FROM
songs JOIN artists ON
songs.artist_id = artists.artist_id
WHERE artists.name =
"Drake"
```

```
SELECT name
FROM songs
WHERE name LIKE "%feat.%"
```

LAB

Find titles of all movies w/ Johnny Depp & Helena Bonham Carter

```
SELECT title FROM movies WHERE id in
  (
    Query to find ids w/ Helena Bonham
                        Carter
  )
INTERSECT
  (
    Query to find
    IDs for Depp
  )
```

Creating Table

```
CREATE TABLE table_name (
name of column  type  Settings,
name of column  type  Settings,
:
)
```

```
CREATE TABLE Students (
person_id INTEGER NOT NULL,
course_id INTEGER NOT NULL);
```

```
CREATE TABLE people (
id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT NOT NULL);
```

```
CREATE TABLE courses (
id INTEGER PRIMARY KEY AUTOINCREMENT,
code TEXT NOT NULL,
title TEXT NOT NULL);
```

LAB

```
INSERT INTO people (name) VALUES ("Luca");
INSERT INTO courses (code, title) VALUES ("CSSO", "Intro CS");
INSERT INTO students (person_id, course_id) VALUES (1,1);
```

id	name
1	Luca

person id	course id
1	1

id	code	title
1	CSSO	Intro CS