SQL

CREATE TABLE celebs **(id INTEGER**, name **TEXT,** age **INTEGER);**
INSERT INTO celebs (id, name, age) **VALUES** (1, 'Justin Bieber', 21);
ALTER TABLE celebs **ADD COLUMN** twitter_handle **TEXT**;
UPDATE celebs **SET** twitter_handle = '@taylorswift13' **WHERE** id = 4;
DELETE FROM celebs **WHERE** twitter_handle **IS NULL**;

- *SQL* is a programming language designed to manipulate and manage data stored in relational databases.
  - A *relational database* is a database that organizes information into one or more tables.
  - A *table* is a collection of data organized into rows and columns.
- A *statement* is a string of characters that the database recognizes as a valid command.
  - CREATE TABLE creates a new table.
  - INSERT INTO adds a new row to a table.
  - SELECT queries data from a table.
  - UPDATE edits a row in a table.
  - ALTER TABLE changes an existing table.
  - DELETE FROM deletes rows from a table.

SELECT DISTINCT genre **FROM** movies;
SELECT * FROM movies **WHERE** imdb_rating > 8;

- = equals
- != not equals
- > greater than
- < less than
- >= greater than or equal to
- <= less than or equal to
-

SELECT * FROM movies
WHERE name **LIKE** '%man%';
SELECT * FROM movies **WHERE** genre = 'comedy' **OR** year < 1980;
SELECT * FROM movies **ORDER BY** imdb_rating **ASC LIMIT** 3; // returns the three lowest rated movies

- SELECT is the clause you use every time you want to query information from a database.

- WHERE is a popular command that lets you filter the results of the query based on conditions that you specify.

- LIKE and BETWEEN are special operators that can be used in a WHERE clause

- AND and OR are special operators that you can use with WHERE to filter the query on two or more conditions.

- ORDER BY lets you sort the results of the query in either ascending or descending order.

- LIMIT lets you specify the maximum number of rows that the query will return. This is especially important in large tables that have thousands or even millions of rows.

SELECT price, COUNT(*) FROM fake_apps **WHERE** downloads > 20000 **GROUP BY** price; **//** Count the total number of apps at each price that have been downloaded more than 20,000 times

SELECT SUM(downloads**) FROM** fake_apps; // returns the total of al doawnloads
SELECT name, category, **MAX(downloads) FROM** fake_apps **GROUP BY** category; // Return the names of the most downloaded apps in each category.

SELECT price, ROUND(AVG(downloads)) FROM fake_apps GROUP BY price; // Calculate the average number of downloads rounded by two decimals at each price

- *Aggregate functions* combine multiple rows together to form a single value of more meaningful information.
- COUNT takes the name of a column(s) as an argument and counts the number of rows where the value(s) is not NULL.
- GROUP BY is a clause used with aggregate functions to combine data from one or more columns.
- SUM() takes the column name as an argument and returns the sum of all the values in that column.
- MAX() takes the column name as an argument and returns the largest value in that column.
- MIN() takes the column name as an argument and returns the smallest value in that column.
- AVG() takes a column name as an argument and returns the average value for that column.
- ROUND() takes two arguments, a column name and the number of decimal places to round the values in that column.

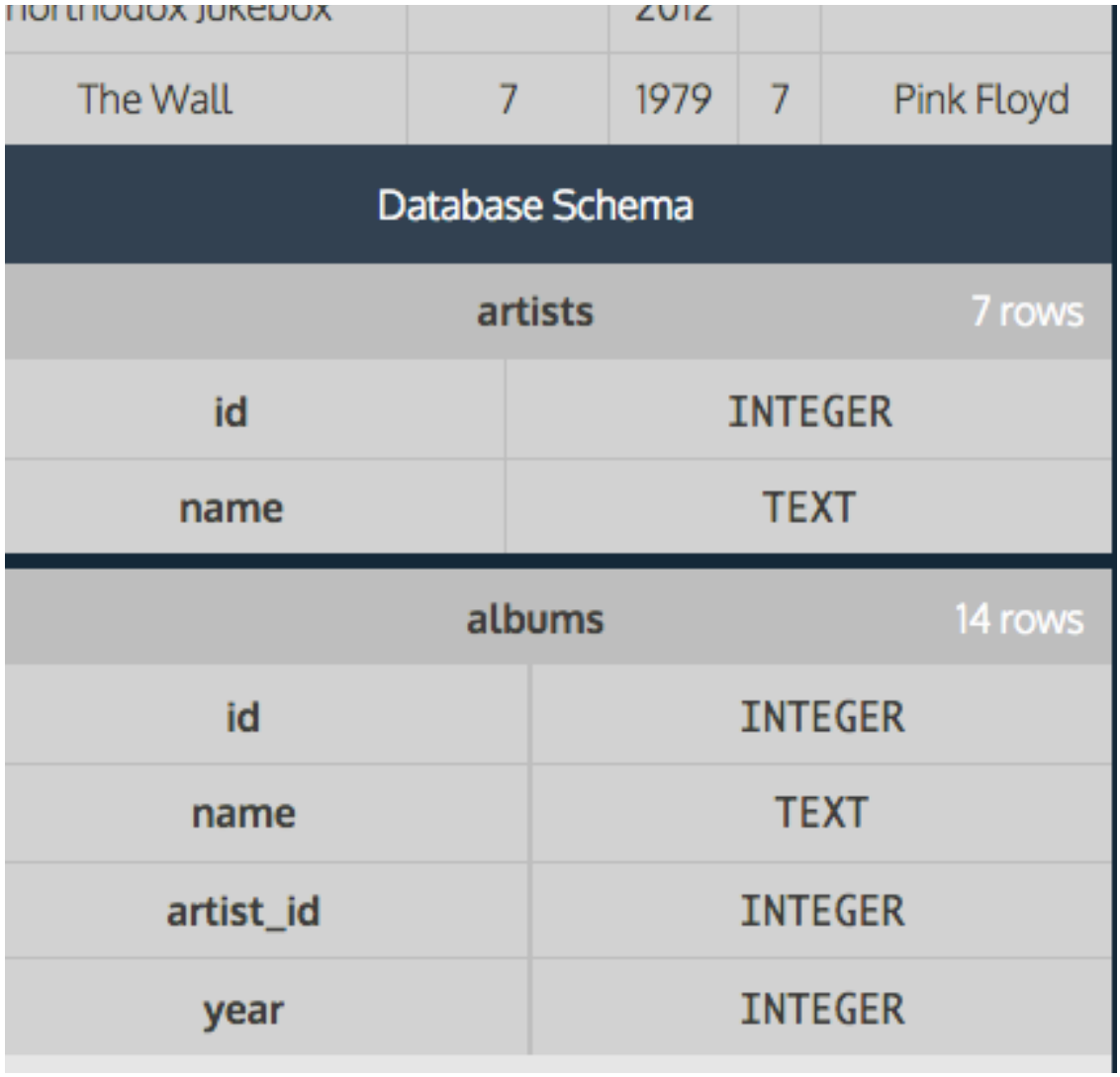
CREATE TABLE artists(id INTEGER PRIMARY KEY, name TEXT);
SELECT * FROM albums WHERE **artist_id** = 3;

SELECT * FROM artists WHERE **id** = 3;
A ***foreign key*** is a **column** that contains **the primary key** of another table in the database. We use **foreign keys** and **primary keys** to **connect rows** in two different tables. One table's foreign key holds the value of another table's primary key. Unlike primary keys, foreign keys do not need to be unique and can be NULL.

SELECT * FROM albums **JOIN** artists **ON** albums.**artist_id =** artists.**id;**
// **a *cross join*.** Here, albums and artists are the different tables we are querying.
When querying more than one table, column names need to be specified by table_name.column_name. Here, the result set includes the name and year columns from the albums table and the name column from the artists table.

SELECT * FROM albums **LEFT JOIN** artists **ON** albums.**artist_id =** artists.**id**;

name	artist_id	year	id	name
A Hard Days Night	1	1964	1	The Beatles
Elvis Presley	2	1956	2	Elvis Presley
1989		2014		
Yellow Submarine	1	1968	1	The Beatles
Hey Jude	1	1970	1	The Beatles
Like a Virgin	4	1984	4	Madonna
om Elvis in Memphis	2	1969	2	Elvis Presley
Bad	3	1987	3	Michael Jackson
Elton John	5	1970	5	Elton John
Like a Prayer	4	1989	4	Madonna
Dark Side of the Moon	7	1973	7	Pink Floyd
Thriller	3	1982	3	Michael Jackson
[illegible]		2012		



Outer joins also combine rows from two or more tables, but unlike inner joins, they do not require the join condition to be met. Instead, every row in the *left* table is returned in the result set, and if the join condition is not met, then NULL values are used to fill in the columns from the *right* table.

The left table is simply the first table that appears in the statement. Here, the left table is albums. Likewise, the right table is the second table that appears. Here, artists is the right table.

```
SELECT
  albums.name AS 'Album',
  albums.year,
  artists.name AS 'Artist'
FROM
  albums JOIN artists ON
  albums.artist_id = artists.id
WHERE
  albums.year > 1980;
```

Query Results		
Album	**year**	**Artist**
Like a Virgin	1984	Madonna
Bad	1987	Michael Jackson
Like a Prayer	1989	Madonna
Thriller	1982	Michael Jackson

Database Schema	
artists	7 rows
id	INTEGER
name	TEXT

albums	14 rows
id	INTEGER
name	TEXT
artist_id	INTEGER
year	INTEGER

- ***Primary Key*** is a column that serves **a unique identifier for row in the table**. Values in this column must be **unique** and **cannot be NULL**.
- ***Foreign Key*** is a **column** that contains the **primary key** to another table in the database. It **is used to identify a particular row** in the referenced table.
- ***Joins*** are used in SQL to combine data from multiple tables.
- **INNER JOIN** will **combine rows** from different tables **if the *join condition* is true**.
- **LEFT OUTER JOIN** will return every row in the ***left*** table, and if the join

condition is not met, NULL values are used to fill in the columns from the *right* table.

- **AS** is a **keyword in SQL** that allows you to **rename a column or table in the result set** using an ***alias***.

```
// replace all occurrences of string1 with string2 for field_name
$sqlinsert    = "UPDATE ".$current_table." SET field_name =
REPLACE(field_name,  'string1',  'string2')";
      $mysqli->query($sqlinsert);
      echo "      Updated: ".$mysqli->affected_rows." on ".$current_table;
      echo "\r\n";
```