

launch _code

LC101 2.7



Class Agenda

1. Last Class Review
2. New Material
3. Studio (Flicklist 4)



Review

Templates

Jinja Variables, Conditionals, Loops

Template Extensions



SQL

SQL - Structured Query Language

It's the base of most querying languages, but there are lots of varieties of SQL: P-SQL, MySQL, SQLite, T-SQL, SQL Server, etc.

We will be using MySQL in this class.



SQL Basics

A SQL Database is a collection of tables, that are connected together via relationships. Each table contains fields in the form of columns, and each new record is it's own row



SQL - Basic Statements

SELECT - Get information from the DB (read)

INSERT - Put information into the DB (create)

UPDATE - Change information in the DB (update)

DELETE - Remove information from the DB
(delete)



Select

The SELECT statement allows us to get information out of the DB. And it usually will look like:

```
SELECT table_column FROM table_name
```

The SELECT statement can't do anything by itself, we have to tell it what we are trying to select. So we have to explicitly tell it which table columns we are looking for, and what table to look for them in.



INSERT INTO

```
INSERT INTO table_name (column_name1,  
column_name2) VALUES (value1, value2)
```

You must have a value for each column you are inserting into.

You don't need to insert into all columns in the table, you can pick and choose as necessary.



UPDATE

UPDATE table_name

SET column_name1 = value1, column_name2 = value2...

WHERE column_name1 = "ID005"

****Without the WHERE condition, you would update all the records in the table****



DELETE

```
DELETE FROM table_name
```

```
WHERE column_name1 = "ID005"
```

****Without the Where condition, you would delete all records from the table. Your table will still exist, along with your shame****



WHERE

WHERE is a conditional statement in SQL. You can select a column that matches the WHERE condition, or update, or delete with the WHERE condition.

```
SELECT column_name1 from table_name WHERE  
column_name2 = 'Paul'
```

This statement will only pull the records of column_name1 from the DB if column_name2 == 'Paul'



Multiple

```
SELECT column_name1, column_name2,  
       column_name3 from table_name1
```

This will pull all the information from columns 1, 2,
and 3 from table_name1

```
SELECT * from table_name1
```

This will pull all the columns, and their associated
data from table_name1

Additional WildCards



CREATE TABLE

```
CREATE TABLE table_name ( column1 datatype,  
column2 datatype, column3 datatype, etc);
```



DROP TABLE

```
DROP TABLE table_name;
```



PK and FK

Primary Key - a unique identifier for a specific record in a specific table.

Table = student

PK = student_ID_number

Foreign Key - a key that relates to a specific record on a different table.

Table = course

PK = course_ID_number

FK = teacher_ID_number



SQL Join Statements

SQL Join statements allow you to join records from multiple tables by utilizing their relationships via PK, FK.

Our prep work mentioned 4 types of joins.

INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN.



JOIN Statement Syntax

SELECT columns

FROM table1

INNER JOIN table2

ON table1.column = table2.column;

This returns records of Table1, and Table2 where they intersect (what they have in common)



One to Many

One Author can write many books.

Author table, has one record with a PK

Our Books table, has many records associated to an author via the FK.




MySQL Data Types

String Datatypes - CHAR, VARCHAR, TEXT, BINARY

Numeric Datatypes - INT, DECIMAL, NUMERIC, FLOAT, BOOL

Date/Time Datatypes - DATE, DATETIME, TIMESTAMP

BLOB (large object) Datatypes - Allocates the size to be multiple bytes, which allows for larger variables, pictures, videos, audio are usually BLOBs.



Studio Walkthrough

We have not installed MySQL on our machines yet, so our first studio (DB Part 1) will be done by practicing writing SQL statements.

As a walkthrough we will create a database (“movie-buff”) that has two tables in it: “movies”, and “directors”.f



Plan our movie table

Movie has:

A Title

A Director

A Release Year

We will also need to be able to uniquely identify this movie so it will need a Movie_ID



Create a Movie table

```
CREATE TABLE movies (  
  movie_id INTEGER PRIMARY KEY  
    AUTO_INCREMENT,  
  title VARCHAR(120),  
  year INTEGER,  
  director VARCHAR(120)  
);
```



Plan our director table

Director has:

A first name

A last name

A country of origin

We will need to be able to uniquely identify each director so let's assign them a director_id



Create Director table

```
CREATE TABLE directors (
```

```
Director_id INTEGER PRIMARY KEY  
AUTO_INCREMENT,
```

```
First VARCHAR(120),
```

```
Last VARCHAR(120),
```

```
Country VARCHAR(120)
```

```
);
```



How are the tables related?

We learned about One to Many relationships. I think that applies here.

One director can direct many movies.

Steven Spielberg directed E.T., and he directed the Indiana Jones Movies, and he directed the Jurassic Park movies, etc.

Director has a one to many relationship with Movies.

• This isn't currently reflected in our DB.



Drop Table

Let's delete our movie table, because we want to create a new movie table that is linked to a director.

```
DROP TABLE movies;
```



CREATE movies table

```
CREATE TABLE movies (
```

```
Movie_id INTEGER PRIMARY KEY AUTO_INCREMENT,
```

```
Title VARCHAR(120),
```

```
Year INTEGER,
```

```
Director_id INTEGER,
```

```
FOREIGN KEY (director_id) REFERENCES  
directors(director_id)
```

```
);
```



Play with the Data

Steven Spielberg (ET 1982)(Jurassic Park 1993)

Martin Scorsese (The Departed 2006)

