# SQL BASICS

```
SELECT * FROM movie;
```

# SQL

- The Structured Query Language is a powerful tool for interfacing with RDBMS's.
- SQL is not the most straightforward language, though, as the syntax is strict, and the wording a little foreign for first time users.
- We will explore SQL by using it to create the Movie database designed earlier, then add and manage some records within the new database.

# CREATE

- The first command to cover is the `CREATE` statement.
- `CREATE` can be used to create a number of different RDBMS entities. A few examples:
  - `CREATE DATABASE`
  - `CREATE TABLE`
  - `CREATE VIEW`
  - `CREATE INDEX`

# CREATE

- We first need to create a database container for our Movie Collection project:
  - `CREATE DATABASE MovieCollection;`
- The name is case sensitive, so we shall use capital letters to denote each word.
- Database naming is a personal or organizational policy decision.  MySQL truly does not care what the name is, so long as it follows the rules laid out in the documentation “Schema Object Names”.  Speaking of…

# DOCUMENTATION! AND LAB!

- Now is a good time to learn about the vast and excellent documentation available for the MySQL system.
  - `http://dev.mysql.com/doc`
- Learn it, Live it, Love it
- Take some time now to peruse the documentation and familiarize yourself with the layout of the site.  Maybe read up on the `mysql` or `mysqladmin` command.
- Find the section on Schema Object Names and browse through it.

# USING A DATABASE

- Now that the MovieCollection database has been created, we need to let the `mysql` client know that is the database we want to work on. This is done with the `mysql` client "`use`" command. But first, let's see what database we're currently using:
  - `SELECT DATABASE();`
- Null? That's not what we want...
  - `use MovieCollection;`
  - `SELECT DATABASE();`

# CREATE TABLE

- With a working database, we're ready to create our tables.
- Creating tables involves naming the table and describing the various column names and types.  Generic syntax:
- ```
  CREATE TABLE table_name (
    column_name  column_type_and_parameters,
    ...
  )
  ```
  ```

# MOVIE TABLE

```
CREATE TABLE movie (

    id          INT UNSIGNED NOT NULL AUTO_INCREMENT,

                PRIMARY KEY ( id ),

    title       VARCHAR(50) NOT NULL,

    acquired    DATETIME,

    borrow_id   INT

);
```

# LAB

1) Create the rest of the tables for the Movie Collection database.

2) Verify all of your tables by comparing to the Movie Collection database schema handout.

3) If you run into problems, ask me for the schema creation handout for guidance.

# INSERT

- Now that we have a database and some tables to work with, let's start adding data! This is accomplished with the `INSERT` statement:
  - `INSERT INTO table_name VALUES ( ... );`
- There are several variations to the `INSERT` statement, so let's take an example insertion and rewrite it in several ways...

# INSERT VARIATIONS

- ```
  INSERT INTO movie
  VALUES ( NULL, 'Hackers', NOW(), NULL );
  ```

- ```
  INSERT INTO movie ( title, acquired )
  VALUES ( 'Hackers', NOW() );
  ```

- ```
  INSERT INTO movie
  SET title='Hackers', acquired=NOW();
  ```
  ```

# LAB

1) Insert the data presented in the handout for the **movie** and **person** tables. You can ignore the other tables for now, as that's too much tedious typing. :)

   Make sure each row goes in successfully, and if you make a mistake, ignore it for now - in a few slides you'll learn how to correct it.

# SELECT

- With data in the database, now the question is how to view it. This is where the `SELECT` statement comes in.
  - `SELECT columns FROM table`
- The `SELECT` statement is a very powerful tool, and there is quite a bit to it. A simple example:
  - `SELECT * FROM movie;`
- This will return all columns for every row in movie

# WHERE CLAUSE

- The bare `SELECT` statement is occasionally useful for a full dump of the information in a table, but normally, we are only interested in specific rows. Filtering the result set in a `SELECT` statement is handled via the `WHERE` clause:
  - `SELECT columns FROM table WHERE conditions`
- Example:
  - `SELECT * FROM person WHERE first = 'John';`

# WHERE CONDITIONS

Column aliases
Basic operators and functions
Booleans
Basic pattern matching

- The conditions for a `WHERE` clause can be simple or extraordinarily complex.
- Let's spend a few minutes on our live systems to experiment with and discover various `SELECT` statements and `WHERE` clauses.

# ORDERING

- `SELECT` statements also support the ability to order results, via the `ORDER BY` clause:
  - `SELECT columns FROM table ORDER BY specification`
- Example:
  - `SELECT * FROM person ORDER BY last ASC;`
- This will return all columns for all rows from the person table ordered by the column ‘last’ in ascending order.

# LIMITING RESULTS

- Sometimes a user is only interested in seeing a subset of the results. This is achieved with the `LIMIT` clause:
  - `SELECT columns FROM table LIMIT number`
- Example:
  - `SELECT * FROM person LIMIT 5;`
- This will return 5 rows from the person table ( no guarantee on which rows are returned! Why? )

# LAB

1) Look through the MySQL documentation on the `SELECT` statement. Can you start to see the power and complexity of this command?

2) How many people in the database have a last name that starts with 'B'? Do this with a proper `SELECT` statement and `WHERE` clause, don't count by hand!

3) Bonus: Create a listing of people where the output is of the form "Last, First Middle". For example: "Jolie, Angelina". Look up the `CONCAT` functions, and possibly `IFNULL`.

# UPDATING DATA

- With the foundation of WHERE clauses, the final two major SQL statements can be explored: UPDATE and DELETE.
- To change information in a table, one makes use of the UPDATE statement:
  - `UPDATE table SET name=value... WHERE conditions`
- Example:
  - `UPDATE movie SET acquired=NOW()`

    `WHERE title = 'Tron';`

# DELETING DATA

- Finally, to remove rows from a table, use the `DELETE` statement:
  - `DELETE FROM table WHERE conditions`
- Example:
  - `DELETE FROM person WHERE last LIKE 'B%';`
- This will delete every row in person where the column 'last' begins with the letter 'B'.

# DELETING STRUCTURES

- When a table, database or any other structure in the RDBMS needs to be completely removed, this is done with the `DROP` statement:
  - `DROP type name`
- Example:
  - `DROP TABLE genre;`
- This will **completely** remove the genre table and all data in it. ***This is irreversible! Be very careful with the* `DROP` *statement!***

# LESS TYPING!

- Until this point, we have been typing every command in manually into the mysql client tool.  This gets old.
- There is a better way!
  - `source filename`
- This will read "`filename`" as a series of SQL commands and run them in the current mysql session.
- A SQL script!

# LAB

1) Drop your MovieCollection database.

2) Go to `http://server1.example.com/mysql`

3) Download "`create-moviedb.sql`" and "`load-moviedb.sql`"

4) Source `create-moviedb.sql`

5) Source `load-moviedb.sql`

6) Perform some queries looking at the data. If you wish to experiment some more with `UPDATE`, `DELETE` and `DROP`, just repeat steps 4 and 5 when you are done.

# JOINS

- Now that we have more data in our tables, we can discuss joins.

- A join occurs when data is pulled from more than one table. The rows from each table are then *joined* together somehow, generally by matching values.

- Consider the actor, movie and person tables. What if we wanted to produce a listing of movie title and actor name. Can it be done?

- Of course!

# JOINS

```
SELECT movie.title AS Title,
CONCAT_WS( '', person.first, ' ', person.last ) AS Name
FROM movie INNER JOIN actor ON
   actor.movie_id = movie.id
INNER JOIN person ON
   actor.person_id = person.id
ORDER BY Title ASC;
```

- Think about that one for a minute...

# JOINS

- The rabbit hole that is joining goes on very, very deep.  There are inner joins, outer joins, left and right joins and more.

- A good book and some experimentation is the best way to master these more advanced topics.  The MovieCollection database, with some additional rows could serve as an excellent learning ground should you be interested.

# LAB

1) Come up with a single SELECT statement which reports each movie title and it's genres.

2) Bonus: come up with a SELECT statement which shows director first and last names and what genre's they have developed.

```
slideshow.end();
```