MySQL

Developers at the beginning of a project. vs. Developers at the end of a project.

Quick aside about uploading files

Don’t believe everything you read on the internet…

https://www.w3schools.com/php/php_file_upload.asp

Vs.

https://secure.php.net/manual/en/function.getimagesize.php

Which should you trust?? ☺

Learning Objectives

Learn and review the basics of database access

Learn how to set up and administer MySQL

Learn how to connect to a database

Adding Database Access

html, JavaScript, PHP let us build user-friendly interfaces for databases

adding a database can make sites more functional and dynamic

- have up to date inventory, user accounts, shopping carts etc.
- dynamically create web pages based on latest data
- save information for later

Database Driven Sites

Many sites are database driven

- Ebay, Amazon, Facebook
- Small mom and pop store fronts
- Photo sharing sites

Steps to connect a website to a db

Set up database

Write programs based on business logic to:

- retrieve
- store
- remove
- update

Create web pages that collect and display db information

Database

Database is a specialized collection of data

Relational database uses tables or *relations* to organize the data

Relational database management system (RDBMS)

- software that allows access to database

Examples of RDBMS:

- DB2, Oracle, MySQL, SQLite, Microsoft Access

Relational Databases

Consists of multiple tables of data (also called *relations*)

Each table is a set of related attributes and possible values

Schema defines the table:

- column headings are the *attributes*
- each value has an associated type

Last	First	Dept.	Email
Carruthers	Sarah	CS	carruths@csci.viu.ca
…			

Relational Databases

Data can be different types

Each row is called a *record* or *tuple*

All records in a table is called *table/relation instance*

A database consists of multiple tables

Relationships between elements in the tables is important

Typically login is required to access data in database

if *localhost* is granted access, then programs running on same host computer can access the database

SQL: Structured Query Language

SQL is a standardized language to create, access and manipulate:

- databases
- tables
- records
- other database-related items

Declarative language

Consists of:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)

SQL

All major RDBMS are SQL compliant (to a degree)

- Makes it easier to work with different databases

However, there are differences between different databases

We will be working with MySQL

Database Queries

Command written in SQL

Retrieval query usually results in a *resultset*

- a table of records

An update query does not return a resultset

Database Queries

Suppose we have *member* table

```
SELECT * FROM member WHERE last='carruthers';
```

- retrieves all columns for all rows where last is carruthers
- resultset is a table of all matches (or a subset of tuples in *member* satisfying the condition in the WHERE clause)

uid	last	first	email	password
jsmith	Smith	Joe	...	...
scarruthers	Carruthers	Sarah	...	...
jjones	Jones	Joel	...	...

Database Queries

Suppose we have *member* table

```
SELECT * FROM member WHERE last='Carruthers';
```

- retrieves all columns for all rows where last is Carruthers
- resultset is a table of all matches (or a subset of tuples in *member* satisfying the condition in the WHERE clause)

uid	last	first	email	password
jsmith	Smith	Joe	...	...
scarruthers	Carruthers	Sarah	...	...
jjones	Jones	Joel	...	...

Database Queries

keywords in SQL are case insensitive

- but all caps is commonly used for clarity

all queries are terminated in ;

* symbol, or *wildcard*, means *all* columns in this query

comments

- start in # to end of line
- start in --SPACE to end of line
- within /* and */ (like C++)

Database Queries

Can also specify which columns to return:

- `SELECT last, first, email FROM member; -- three columns`
- `SELECT password FROM member where uid="scarruthers"; # password for sarah`

uid	last	first	email	password
jsmith	Smith	Joe	…	…
scarruthers	Carruthers	Sarah	…	…
jjones	Jones	Joel	…	…

Database Queries

To avoid duplicate rows, use DISTINCT

- `SELECT DISTINCT zip FROM employee;`
- `SELECT DISTINCT city, state, country FROM participant;`
- `SELECT DISTINCT major, year FROM student WHERE year=“Freshman”;`

Database Queries

Can order the resultset using the ORDER BY clause

- `SELECT * FROM client ORDER BY last_name;`
- `SELECT id, grade FROM student_grade WHERE course_id="CSCI311" AND semester="Spring" AND year="2017" ORDER BY grade;`

Normally ordering is low to high (ascending)

- use DESC keyword to indicate descending
- `SELECT customer_name, amt FROM sale ORDER BY amt DESC;`

MySQL Data types

MySQL has many different data types:

- String Types
- Integer Types
- Floating Point Types
- Fixed Point Types
- Bit Value Types
- Numeric type Attributes
- Date and time Types
- Blob and Text Types
- Enum Types
- Set Types

String types

Type	Description
CHAR	non-binary strings Fixed length right-padded to specified length up to 255 characters
VARCHAR	non-binary strings variable-length strings up to 255 characters
TINYTEXT TEXT MEDIUMTEXT LONGTEXT	string with up to 255 characters string with up to 65535 characters string with up to 16777215 characters string with up to 4,294,967,295 characters

Integer Types

Type	Length (bytes)	Minimum Value signed unsigned	Maximum Value signed unsigned
TINYINT	1	-128 0	127 255
SMALLINT	2	-32768 0	32767 65535
MEDIUMINT	3	-8388698 0	8388607 16777215
INT	4	-2147483648 0	2147483647 4294967295
BIGINT	8	-92233720368547758 08 0	9223372036854775807 18446744073709551615

Floating and fixed point types

Type	Description
FLOAT	4-byte single precision floating point number precision from 0 to 23
DOUBLE	8-byte double precision floating point number precision from 24 to 53
DECIMAL NUMERIC	A fixed point number Maximum number of digits: 65 (pre MySQL 5.03) or 64 (post MySQL 5.04) stored as a string

Examples

```
CREATE TABLE course(
    name VARCHAR(40),
    courseID VARCHAR(8),
    level INT(1) );
INSERT INTO course(name, courseID, level)
VALUES ('Data Structures', 'CSCI260', 2),
('System and Networks', 'CSCI251', 2);
```

Examples

```
CREATE TABLE course(
    name VARCHAR(40),
    courseID VARCHAR(8),
    level INT(1) );
```

name	courseID	level
”Data Structures”	“CSCI260”	2
“System and Networks”	“CSCI251”	2

```
INSERT INTO course(name, courseID, level)
VALUES ('Data Structures', 'CSCI260', 2),
('System and Networks', 'CSCI251', 2);
```

Date and time types

Type	Description	Display Format	Range
DATETIME	A date and time combo	YYYY-MM-DD HH:MI:SS	1000-01-01 00:00:00 to 9999-12-31 23:59:59
DATE	Date only	YYYY-MM-DD	1000-01-01 to 9999-12-31
TIMESTAMP	timestamp value stored as the number of seconds since 1970 UTC	YYYY-MM-DD HH:MI:SS	1970-01-01 00:00:00 UTC to 2038-01-09 03:14:07 UTC
TIME	Time only	HH:MI:SS	-838:59:59 to 838:59:59
YEAR	Year only in 2 or 4 digit format	YYYY or YY	1901 to 2155 (4 digit) 70 to 69 (1970 to 2069)

Blob Types

Type	Description
BLOB	Binary Large Object Can contain variable amount of data Treated as binary strings up to 65535 bytes of data
MEDIUMBLOB	up to 16777215 of data
LONGBLOB	up to 4294967295 of data

And the rest…

Type	Description
ENUM	string object with a list of possible values up to 65535 values Cannot add values that are not in list sorted in the order they are entered
SET	String object with up to 64 comma separated values

Some Examples

```
CREATE TABLE shirts (
        name VARCHAR(40),
        size ENUM('x-small', 'small', 'medium', 'large', 'x-large') );
INSERT INTO shirts (name, size) VALUES ('dress shirt','large'), ('t-shirt','medium'), ('polo shirt','small');
SELECT name, size FROM shirts WHERE size = 'medium';
+---------+--------+
| name    | size   |
+---------+--------+
| t-shirt | medium |
+---------+--------+
```

MySQL Expressions

in the WHERE clause the expression after the WHERE keyword is evaluated

- if true or 1 the clause is satisfied
- if false or 0 the clause is not satisfied

Can include relational operators:

- = equal
- <>, != not equal
- >, < greater than, less than
- >=, <= greater than or equal, less than or equal

MySQL TRUE, FALSE, OTHER…

MySQL relational operators evaluate to TRUE, FALSE or NULL

0, ‘’, and NULL treated as logical false

everything else is true

To test for null, use IS NULL or IS NOT NULL

<=> operator: null-safe returns 1 if both operands are NULL, otherwise works like = operator

MySQL logical operators

MySQL has the following logical operators:

- AND
- OR
- NOT
- XOR

- 0 AND NULL is 0
- 1 AND NULL is NULL
- 0 OR NULL is NULL
- 1 OR NULL is 1
- 1 XOR NULL is NULL
- 0 XOR NULL is NULL
- NOT NULL is NULL

Mathematical Operators

MySQL has typical:

- + * - /
- / usually return a floating point value,
- DIV operator does integer division
- divide by 0 returns NULL

- = operator overloaded to also be assignment when in the SET environment

MySQL SELECT Queries

We can perform pattern matching in WHERE clause

- LIKE
- NOT LIKE
- % matches any string with 0 or more characters
- _ matches every single character
- \ is escape character

Examples:

- `SELECT email FROM member WHERE zip LIKE “44%”`
  - returns email of records where zip code matches all zip codes that start with 44
- `SELECT * FROM student WHERE phone NOT LIKE ‘333-%’`
  - returns all records where phone doesn’t start with 333-

MySQL pattern matching

string comparisons in MySQL are *usually* case insensitive

some operands use case-sensitive *collating sequences*

To force case-sensitive comparison:

- `SELECT “ABC” < “abc” COLLATE utf8_bin; -- returns 1`
- `SELECT filename LIKE ‘%.html’ COLLATE utf8_bin;`
  - /* returns all filenames that end in .html */

Can also use regular expressions

- `filename RLIKE ‘/.html*’ -- foo.html or bar.htm`
- `filename RLIKE ‘/.jpg$|/.JPG$’ -- foo.jpg or bar.JPG`

Producing new columns

A SELECT can make new columns using the AS clause:

- `SELECT name, vacation_taken, vacation_accrued, (vacation_accrued-vacation_taken) AS vacation_balance FROM employee;`
- `SELECT CONCAT(last,', ', first) AS fullname FROM member ORDER BY fullname;`

For more functions:
https://dev.mysql.com/doc/refman/5.7/en/functions.html

Aggregating Attribute Values

It can be useful to calculate or aggregate data rather than just returning another table of data

MySQL built-in aggregating functions:

- COUNT(expr) returns count of non-null values of *expr*. count(*) returns number of rows in resultset
- AVG(expr) returns average of expr values
- MAX(expr) returns max of expr values
- MIN(expr) returns min of expr values
- SUM(expr) returns sum of expr values
- GROUP_CONCAT(expr) returns the comma-separated string concatenation of expr values

Aggregating examples

```
SELECT COUNT(*) AS enrollment FROM student;
```

- resultset: single record with enrollment with a value of 4

```
SELECT COUNT(letter_grade) FROM grade WHERE
letter_grade='A';
```

- resultset: number of A's

```
SELECT COUNT(DISTINCT major) FROM student
```

- how many different majors

```
SELECT AVG(hw2) as hw2_avg, MAX(hw2) as hw2_max,
MIN(hw2) as hw2_min FROM grade;
```

Aggregating examples

```
SELECT dept_name, COUNT(*) AS enrollment FROM student GROUP BY dept_name ORDER BY enrollment DESC
```

- aggregates over a group of rows in a table
- resultset is a table with 2 columns: dept_name and enrollment
- Each row will have a department, and the number of students enrolled

Data Relationships

SQL databases get their power from *relationships* between data

Instead of large complicated tables, we separate common concepts into different tables

Relationships can be:

- 1-1
- many-1
- many-many
- ...

Tables are *related* using keys

- Primary Keys
- Foreign Keys

Normalization

To eliminate duplicated information we *normalize* our data

There are different levels of normalization:

- First Normal Form (1NF)
- …
- Fifth Normal Form (5NF)

Zero Normal Form

This is just a table of data

Each record (row) is self-contained

- doesn’t need to reference anything else

SONG TITLE	ARTIST	GENRE	SUB-GENRE	YEAR
Shannon	Henry Gross	Rock	Light Rock	1976
Lover's Will	Bonnie Raitt	Rock	Light Rock	1998
I Don't Wanna Live Without Your Love	Chaptercago	Rock	Light Rock	1988
Heart Attack	Olivia Newton-John	Pop	Adult Contemporary	1982
In A Dream	Badlands	Rock	Hard Rock	1991
With A Little Luck	Paul McCartney	Rock	Classic Rock	1978
It's A Miracle	Barry Manilow	Pop	Adult Contemporary	1975
It's Only Love	Bryan Adams / Tina Turner	Pop	Adult Contemporary	1984
Jazzman	Carole King	Pop	Adult Contemporary	1974
Jesse	Carly Simon	Pop	Adult Contemporary	1980
Just Like Jesse James	Chapterr	Pop	Adult Contemporary	1989
Little Miss Cannot Be Wrong	Spin Doctors	Pop	Adult Contemporary	1992
Lost In Love	Air Supply	Pop	Adult Contemporary	1980
Good Times	Sam Cooke	Hip-Hop	Soul	1964
Make It With You	Bread	Pop	Adult Contemporary	1970
Mandy	Barry Manilow	Pop	Adult Contemporary	1974
Miss Chaptertelaine	K.D. Lang	Pop	Adult Contemporary	1992

First Normal Form

Create separate tables for related information

Eliminate duplicated columns

Create primary keys for each table

Going to First Normal Form

Band name

Album title

Song titles

Song length

Producer Name

Release Year

Artist hometown

Concert Venue

Concert Date

Artist Names

Going to First Normal Form

~~Band name~~

Album title

Song titles

Song length

Producer Name

Release Year

Artist hometown

Concert Venue

Concert Date

Artist Names

Bands Band Name		

Going to First Normal Form

~~Band name~~

~~Album title~~

Song titles

Song length

Producer Name

Release Year

Artist hometown

Concert Venue

Concert Date

Artist Names

Bands	**Albums**	
Band Name	Album Name (ref Band)	

Going to First Normal Form

~~Band name~~

~~Album title~~

~~Song titles~~

~~Song length~~

Producer Name

Release Year

Artist hometown

Concert Venue

Concert Date

Artist Names

Bands	**Albums**	**Songs**
Band Name	Album Name (ref Band)	Song Title Song Length (ref Album)

Going to First Normal Form

~~Band name~~

~~Album title~~

~~Song titles~~

~~Song length~~

~~Producer Name~~

~~Release Year~~

~~Artist hometown~~

~~Concert Venue~~

~~Concert Date~~

~~Artist Names~~

Bands BandID Band Name	**Albums** AlbumID Album Name Release Year (ref Band)	**Songs** SongID Song Title Song Length (ref Album)
Labels ProducerID Producer Name	**Artists** ArtistID Artist Name Hometown	**Concerts** VenueID Venue Date

Second Normal Form

Is in First Normal Form

Move repeated data to reference table

Connect reference tables using foreign keys

Going to Second Normal Form

Bands BandID Band Name	**Albums** AlbumID Album Name Release Year (ref Band)	**Songs** SongID Song Title Song Length (ref Album)
Labels ProducerID Producer Name	**Artists** ArtistID Artist Name Hometown	**Concerts** VenueID Venue Date

Going to Second Normal Form

Bands BandID Band Name	**Albums** AlbumID Album Name Release Year BandID ProducerID	**Songs** SongID Song Title Song Length AlbumID	
Labels ProducerID Producer Name	**Artists** ArtistID Artist Name Hometown	**Concerts** VenueID Venue Date	**Bands2Labels** id producerID bandID timestamp

MySQL CRUD Actions

We can break down everything we do into:

- Create
- Read
- Update
- Delete

Opening MySQL

From a Browser (using PHPMyAdmin)

- typically found at: localhost/phpmyadmin (and BAD SECURITY-wise!)

From the command line

- mysql -h wwwstu.csci.viu.ca -p (for CSCI install)
- see what's there:
  - show databases;
- creating databases:
  - create database NAME;
- drop a database:
  - drop database NAME;
- use a database:
  - use music;

Some Examples

Open MySQL

View databases: `show databases;`

```
+--------------------+
| Database           |
+--------------------+
| carruths           |
| information_schema |
+--------------------+
2 rows in set (0.00 sec)
```

Some Examples

Select database: `use databasename;`

show tables: `show tables;`

```
+----------------------+
| Tables_in_carruths   |
+----------------------+
| Furniture            |
| albums               |
| bands                |
| directors            |
| testtable            |
| tv_series            |
| tv_series_directors  |
+----------------------+
7 rows in set (0.00 sec)
```

Some Examples

View contents of a table: `SELECT * FROM Furniture;`

prod_number	name	date_added	category	type	description	price	pix
1	Fluffy Chair	2017-01-30	Living Room	Chair	very fluffy chair	43.75	fluffy.jpg
2	Hard Chair	2017-01-30	Dining Room	Chair	very hard chair	13.75	hardchair.jpg
3	Hipsteryer Table	2017-01-30	Dining Room	Table	artisan table	743.75	hipster.jpg
4	Hipster Table	2017-01-30	Dining Room	Table	artisan table	743.75	hipster.jpg

Some Examples

Insert data into table: `INSERT INTO Furniture (name, date_added, category, type, description, price) VALUES ("Table Lamp", NOW(), "Living Room", "Lamp", "A classy lamp", 149.99);`

prod_number	name	date_added	category	type	description	price	pix
1	Fluffy Chair	2017-01-30	Living Room	Chair	very fluffy chair	43.75	fluffy.jpg
2	Hard Chair	2017-01-30	Dining Room	Chair	very hard chair	13.75	hardchair.jpg
4	Hipster Table	2017-01-30	Dining Room	Table	artisan table	743.75	hipster.jpg
11	Table Lamp	2018-03-07	Living Room	Lamp	A classy lamp	149.99	missing.jpg

Create

Once we've created a database, we need to add tables to it

the music database already contains the band and albums tables

if they weren't there we could create them:

```
CREATE TABLE bands (bandID int not null auto_increment primary
key, bandName varchar(40) not null);
```

commas separate the column definitions

bandID:

- is an int, cannot be empty, created automagically, and is the primary key

bandName:

- a var char of length 40, and not empty

Create

We can also insert data into our tables

- using the INSERT query

```
INSERT INTO bands (bandName) values (“Michael Jackson”), (“Prince”);
```

And we can see what is in the table:

```
SELECT * FROM bands;
```

Read

Once we have data in our database we can use Select statements to read the data:

- Can get one or more records
- Can get one or more columns of these records
- We can make new columns

Read

Get all data:

```
   Select * from bands

+--------+--------------+

| bandID | bandName     |

+--------+--------------+

|      1 | The Who      |

|      2 | Moxy Fruvous |

|      3 | The Doors    |

|      4 | Maroon 5     |

+--------+--------------+

4 rows in set (0.00 sec)
```

Read

Get specific columns:

```
Select albumName, releaseDate from bands
+---------------+--------------+
| albumName     | releaseDate |
+---------------+--------------+
| Tommy         | 1969-05-23  |
| Bargainville | 1993-07-20  |
| Full Circle  | 1972-07-17  |
+---------------+--------------+
3 rows in set (0.00 sec)
```

Read

We can rename the columns:

```
select albumName as "Album", releaseDate as "Release Date" from albums;
+---------------+--------------+
| Album         | Release Date |
+---------------+--------------+
| Tommy         | 1969-05-23   |
| Bargainville | 1993-07-20   |
| Full Circle   | 1972-07-17   |
+---------------+--------------+
3 rows in set (0.00 sec)
```

Read

We can retrieve only specific records:

```
select albumName as "Albums", bandID as "Band"
from albums where albumName like '%bargain%';
+---------------+-------+
| Albums        | Band |
+---------------+-------+
| Bargainville |    2 |
+---------------+-------+
1 row in set (0.00 sec)
```

Read

We can sort as well:

```
select albumName as "Albums", releaseDate as "Release Date" from albums
order by albumName;
+---------------+--------------+
| Albums        | Release Date |
+---------------+--------------+
| Bargainville | 1993-07-20   |
| Full Circle  | 1972-07-17   |
| Tommy        | 1969-05-23   |
+---------------+--------------+
3 rows in set (0.00 sec)
```

Read

We can aggregate data (say, count the number of albums per bandID

```
select bandID, count(*) as "Albums" from albums group by bandID;
+--------+--------+
| bandID | Albums |
+--------+--------+
|      1 |      1 |
|      2 |      1 |
|      3 |      1 |
|      5 |      2 |
+--------+--------+
4 rows in set (0.00 sec)
```

Read

We can use logic in the where clause to limit the return results

```
select * from albums where albumName like "%full%" or albumName like
"%to%";
+---------+---------------------+-------------+--------+-------------+
| albumID | albumName           | releaseDate | bandID | producerID |
+---------+---------------------+-------------+--------+-------------+
|       1 | Tommy               | 1969-05-23  |      1 |        NULL |
|       3 | Full Circle         | 1972-07-17  |      3 |        NULL |
|       4 | Under the Mistletoe | 2010-11-01  |      5 |        NULL |
+---------+---------------------+-------------+--------+-------------+
3 rows in set (0.00 sec)
```

Read

Or limit the number of rows we care about

```
select * from albums where albumName like "%full%" or albumName
like "%to%" limit 2;

+----------+--------------+--------------+---------+-------------+

| albumID | albumName    | releaseDate | bandID | producerID |

+----------+--------------+--------------+---------+-------------+

|        1 | Tommy        | 1969-05-23  |       1 |        NULL |

|        3 | Full Circle | 1972-07-17  |       3 |        NULL |

+----------+--------------+--------------+---------+-------------+

2 rows in set (0.00 sec)
```

Update

We can also update data in our database

```
create table labels (producerID int not null auto_increment primary key, producer
varchar(40) not null);
Query OK, 0 rows affected (0.02 sec)
mysql> show tables;
+-----------------+
| Tables_in_music |
+-----------------+
| albums          |
| bands           |
| labels          |
+-----------------+
3 rows in set (0.00 sec)
```

Update

Now let’s add a couple of producers:

```
insert into labels (producer) values ("Stewart"), ("Messinger"), ("Moxy Fruvous");
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from labels;
+------------+--------------+
| producerID | producer     |
+------------+--------------+
|          1 | Stewart      |
|          2 | Messinger    |
|          3 | Moxy Fruvous |
+------------+--------------+
3 rows in set (0.00 sec)
```

Update

Now we're ready to link...

```
update albums set producerID=1 where albumName="Under the Mistletoe";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> select * from albums;
+---------+---------------------+-------------+--------+------------+
| albumID | albumName           | releaseDate | bandID | producerID |
+---------+---------------------+-------------+--------+------------+
|       1 | Tommy               | 1969-05-23  |      1 |       NULL |
|       2 | Bargainville        | 1993-07-20  |      2 |       NULL |
|       3 | Full Circle         | 1972-07-17  |      3 |       NULL |
|       4 | Under the Mistletoe | 2010-11-01  |      5 |          1 |
|       5 | Believe             | 2012-06-15  |      5 |       NULL |
+---------+---------------------+-------------+--------+------------+
5 rows in set (0.00 sec)
```

Update

Now we're ready to link...

```
select * from albums;
+---------+---------------------+-------------+--------+-------------+
| albumID | albumName           | releaseDate | bandID | producerID |
+---------+---------------------+-------------+--------+-------------+
|       1 | Tommy               | 1969-05-23  |      1 |       NULL |
|       2 | Bargainville        | 1993-07-20  |      2 |          3 |
|       3 | Full Circle         | 1972-07-17  |      3 |       NULL |
|       4 | Under the Mistletoe | 2010-11-01  |      5 |          1 |
|       5 | Believe             | 2012-06-15  |      5 |          2 |
+---------+---------------------+-------------+--------+-------------+
5 rows in set (0.00 sec)
```

Delete

Finally, we may need to remove records from the database's tables

```
delete from albums where albumName like “%like%”;
```

More advanced queries

There are other more advanced ways to retrieve data from our database:

- joins
- nested queries

Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14
07-Jan	**14-Jan**	**22-Jan**	**28-Jan**	**04-Feb**	**11-Feb**	**18-Feb**	**25-Feb**	**04-Mar**	**11-Mar**	**18-Mar**	**25-Mar**	**01-Apr**	**08-Apr**
Overview Admin. Intro to the Web	HTML CSS	HTML forms Accessibility	JavaScript I	PHP I	JavaScript II Midterm I	Responsive Design	No Classes	PHP II and MySQL	MySQL	Security Midterm II	Security	jQuery AJAX	Review
Quiz 1: Web Quiz 2: HTML	Quiz 3: CSS	Quiz 4: Forms	Quiz 5: JS I	Quiz 6: PHP I	Quiz 7: JS II	Quiz 8: Responsive Design		Quiz 9: PHP II	Quiz 10: MySQL	Quiz 11: Security	Quiz 12: jQuery & AJAX		
No labs	**Lab 1**	**Lab 2**	**Lab 3**	**Lab 4**	**Lab 5**	**Lab 6**	**Lab 7**	**Lab 8**	**Lab 9**	**Lab 10**	**Lab 11**	**Lab 12**	**Lab 12**
Labs	Labs and Tools	HTML and CSS	HTML and Forms	JavaScript I	PHP I	Responsive Design	JavaScript II	PHP II	MySQL I	Security	The whole Shebang	Project demos	
Project deliverables					Proposal Due Roles and Resp. Due		Prototype Due		Midterm Assessment Due		Project due		Final Assessment Due

Joins

Different ways to join tables:

- Inner join:
  - return records from 2 tables when a value is found in both
- Left/right join:
  - return records from 2 tables when a value is NOT found in both

Inner Join

lets get all records (from both the bands and albums tables) that have an album

- that is, there is an entry in the album table that matches the id of a band

```
select * from bands, albums where albums.bandID=bands.bandID;
+--------+---------------+---------+---------------------+-------------+--------+------------+
| bandID | bandName      | albumID | albumName           | releaseDate | bandID | producerID |
+--------+---------------+---------+---------------------+-------------+--------+------------+
|      1 | The Who       |       1 | Tommy               | 1969-05-23  |      1 |       NULL |
|      2 | Moxy Fruvous  |       2 | Bargainville        | 1993-07-20  |      2 |          3 |
|      3 | The Doors     |       3 | Full Circle         | 1972-07-17  |      3 |       NULL |
|      5 | Justin Bieber |       4 | Under the Mistletoe | 2010-11-01  |      5 |          1 |
|      5 | Justin Bieber |       5 | Believe             | 2012-06-15  |      5 |          2 |
+--------+---------------+---------+---------------------+-------------+--------+------------+
5 rows in set (0.00 sec)
```

Inner Join

We can make the output nicer:

```
select bands.bandName as "Bands", albums.albumName as "Album", releaseDate as
"Released" from bands, albums where albums.bandID=bands.bandID;
+---------------+---------------------+------------+
| Bands         | Album               | Released   |
+---------------+---------------------+------------+
| The Who       | Tommy               | 1969-05-23 |
| Moxy Fruvous  | Bargainville        | 1993-07-20 |
| The Doors     | Full Circle         | 1972-07-17 |
| Justin Bieber | Under the Mistletoe | 2010-11-01 |
| Justin Bieber | Believe             | 2012-06-15 |
+---------------+---------------------+------------+
5 rows in set (0.00 sec)
```

Left/right joins

So far we haven’t seen Maroon 5 in these results, because they don’t have an album

We can use left/right joins to get rows like this

- Left join: all records from left table, plus values from right if they exist
- Right join: all records from right table, plus values from left if they exist

Left/right joins

Lets get all albums with their producer with left join

```
select albums.albumName as "Albums", labels.producer as "Producer" from
albums left join labels on albums.producerID=labels.producerID;
+----------------------+---------------+
| Albums               | Producer      |
+----------------------+---------------+
| Under the Mistletoe | Stewart       |
| Believe              | Messinger     |
| Bargainville         | Moxy Fruvous |
| Tommy                | NULL          |
| Full Circle          | NULL          |
+----------------------+---------------+
5 rows in set (0.00 sec)
```

Left/right joins

Do the same as a right join

```
select albums.albumName as "Albums", labels.producer as "Producer"
from albums right join labels on
albums.producerID=labels.producerID;
+----------------------+---------------+
| Albums               | Producer      |
+----------------------+---------------+
| Bargainville         | Moxy Fruvous  |
| Under the Mistletoe  | Stewart       |
| Believe              | Messinger     |
+----------------------+---------------+
3 rows in set (0.00 sec)
```

joins

```
select bandName, albumName, releaseDate, producer from bands, albums,
labels where albums.bandID=bands.bandID and
albums.producerID=labels.producerID;
+----------------+----------------------+--------------+--------------+
| bandName       | albumName            | releaseDate | producer     |
+----------------+----------------------+--------------+--------------+
| Moxy Fruvous   | Bargainville         | 1993-07-20   | Moxy Fruvous |
| Justin Bieber | Under the Mistletoe | 2010-11-01   | Stewart      |
| Justin Bieber | Believe              | 2012-06-15   | Messinger    |
+----------------+----------------------+--------------+--------------+
3 rows in set (0.00 sec)
```

joins

```
select bandName, albumName, releaseDate, title, length, producer from bands,
albums, songs, labels where songs.albumID=albums.albumID and
albums.bandID=bands.bandID and albums.producerID=labels.producerID;

+---------------+-----------+-------------+-----------+--------+-----------+

| bandName      | albumName | releaseDate | title     | length | producer  |

+---------------+-----------+-------------+-----------+--------+-----------+

| Justin Bieber | Believe   | 2012-06-15  | Boyfriend |    171 | Messinger |

| Justin Bieber | Believe   | 2012-06-15  | Take You  |    220 | Messinger |

+---------------+-----------+-------------+-----------+--------+-----------+

2 rows in set (0.00 sec)
```

Nested queries

Nested queries let us use the results of one query *in* another query

For example, say we want to find info about the shortest song:

```
SELECT title, length FROM songs WHERE length=(SELECT
MIN(length) FROM songs);
+------------+---------+
| title      | length |
+------------+---------+
| Boyfriend |    171 |
+------------+---------+
1 row in set (0.00 sec)
```

Nested queries

We can combine the above with a join to get more data

```
select bandName, title, albumName from bands, albums, songs
where songs.length=(select min(length) from songs) and
songs.albumID=albums.albumID and
albums.bandID=bands.bandID;
+----------------+------------+------------+
| bandName       | title      | albumName  |
+----------------+------------+------------+
| Justin Bieber | Boyfriend | Believe    |
+----------------+------------+------------+
1 row in set (0.00 sec)
```

Nested Queries

other ways to search in the where clause:

- ANY, IN, SOME, ALL, EXISTS

```
select bandName from bands where bands.bandID = ANY (select
bandID from albums where albums.albumName like "%ll%");
+---------------+
| bandName      |
+---------------+
| Moxy Fruvous |
| The Doors     |
+---------------+
```

Accessing databases from php

PHP can connect to a MySQL database

- MySQLi extension
- Using PHP Data Objects (PDO) (https://secure.php.net/manual/en/class.pdo.php)

Which to choose?

- They are functionally equivalent, but MySQLi only works with MySQL
- Both are OO
- Both support Prepared Statements
  - helps protect against SQL Injection
- PDO works with multiple databases
  - better if you may need to access other databases

PDO connection

First step is to connect to the database

Create an instance of a PDO:

- $dbh = new PDO(“mysql:host=HOSTNAME;dbname=DATABASENAME”, USERNAME, PASSWORD);
- it is best to keep the values (HOSTNAME, etc.) in a separate file
- should also put the connection call in a try catch to handle exceptions

```
<html>
<head><title>Database connection test</title></head>
<body>
<h1>Trying to connect...</h1>
<p><?php
$servername = "localhost";
$username = "carruths";
$password = "password";
$database = "carruths";
try{
	$dbh = new PDO("mysql:host=$servername;dbname=$database", $username,
$password);
	echo "Connect successfully";
}catch(PDOException $e){
	echo "Connection failed: " . $e->getMessage();

}
?></p>
</body></html>
```

Querying the database

Create a query

pass it as a parameter to the query method

iterate through each record

```
<!--
<html>
<head><title>Database connection test</title></head>
<body>
<h1>Trying to connect...</h1>
<p><?php
$servername = "localhost";
$username = "carruths";
$password = "password";
$database = "carruths";
try{
        $dbh = new PDO("mysql:host=$servername;dbname=$database", $username, $password);
        $myQuery = "SELECT * FROM bands";
        $resultset = $dbh->query($myQuery);
        foreach ($resultset as $row){
                echo "<pre>";
                print_r($row);
                echo "</pre>";
        }

}catch(PDOException $e){
        echo "Connection failed: " . $e->getMessage();

}
?></p>
</body></html>
```

Inserting into the database

Build a query using Insert

Because no resultset is returned, we use the exec function

if the insert fails, exec will return false

```
<html>
<head><title>Database connection test</title></head>
<body>
<h1>Trying to connect...</h1>
<p><?php
$servername = "localhost";
$username = "carruths";
$password = "password";
$database = "carruths";
try{
        $dbh = new PDO("mysql:host=$servername;dbname=$database", $username, $password);
        echo "Connect successfully\n";
        $myQuery = "insert into albums (albumName, bandID, releaseDate) values ('Strange
Days', 3, '1967-10-16')";
        if($dbh->exec($myQuery) !==false){
                echo "The album was inserted";
        }else{
                echo "The album was not inserted";
        }
        $dbh = null;
}catch(PDOException $e){echo "Connection failed: " . $e->getMessage();}
?></p>
</body></html>
```

Insert using a form

See insertForm.php

Use the value of the dropdown for bandID

Let user type in an album

Add a value, and see it in db

More advanced example

Let’s build a table that contains the contents of the db

One album per row

Use the resultset to fill the table (eventually)

iterate through each row of the resultset

- and for each row, iterate through the values in it:

```
foreach($row as $field => value){
    echo “field: $field, and value: $value <br”;
}
```

Using data from 2 tables

Now let’s use the band table to get the band’s name

Use a join to create a resultset with all the columns of both

- yes, we could be more specific...

Note about password information etc.

put connection information in a separate file and include it where needed

give the file the extension .inc (not php) to minimize risk of it being run as php by accident

give it the following permissions: 600

- should not be readable by anyone but owner

Prepared statements

When we need to execute similar statements over and over

- more efficient to use “prepared” statements
- reduces parsing time
- minimizes bandwidth
- help prevent SQL injections

How does it work?

- create a `prepare` statement that works like a template
- later, when we want to use it, values are binded to the parameters

More involved examples

Build a table using a join query (musicDemo.php)

Add paging to the table (musicPaginateDemo.php)

Add a launching page where user can select band from list, then see all their songs (showAlbums.php)

Summary

We've seen basic MySQL functionality

Syntax

Operators

Basic Queries

More advanced queries

Accessing database from PHP

- connections
- forming queries
- working with data returned