**SQL Library**

**MySQL**

**Data Types**

* CHAR[(size)]
  + fixed-length string
  + default size is 1 & max is 255
* VARCHAR(size)
  + variable-length string of maximum size
* TEXT
  + variable-length characters up to 65,535 bytes
* INT[(size)]
  + integer (whole number) values of various length
* TINYINT[(size)]
  + integer between -127 & 128 (default size is 4)
* DECIMAL[(p[,s])]
  + decimal values that can contain an integer portion and a decimal portion
  + p is the precision (total # digits) and s is the scale (# of decimal places)
* DATE, TIME, or DATETIME
  + DATE
    - up through year 9999
    - default format is “yyyy-mm-dd”
  + TIME
    - default format is “hh:mi:ss”
  + DATETIME
    - up through year 9999
    - default format is “yyyy-mm-dd hh:mi:ss”

**Referential Integrity**

* Declerative Referential Integrity (DRI)
  + Use InnoDB tables to enforce referential integrity
  + Define foreign key restraints

Creating Indexes

* Foreign keys
* When column is frequently used in queries/joins
* Contains large number of distinct values
* Not updated frequently

**Data Definition Language (DDL)**

* Check if something exists (modifiers to be included between action and object if desired)
  + IF NOT EXISTS
  + IF EXISTS
* Create a database
  + CREATE DATABASE database\_name;
    - Throws an error if the database exists
  + CREATE DATABASE IF NOT EXISTS database\_name;
    - Will only execute if the database does not exist
* Selecting a database
  + USE database\_name;
    - Not very common with SQL, but it’s a MySQL extension commonly used there
* Dropping (deleting) items
  + DROP DATABASE IF EXISTS database\_name;
  + DROP TABLE IF EXISTS tableName;
* Creating tables
  + Need a comma at the end of each row of code (except the last row)
  + CREATE TABLE tableName

(

columnName1 dataType [columnAttributes][,

columnName2 dataType [columnAttributes]]…

)

* + Attributes
    - PRIMARY KEY
      * Automatically NOT NULL UNIQUE and indexed
      * Can include as an attribute (column level)
      * Can code at the end of the CREATE TABLE () statement (table level)
        + Only way to specify multiple primary keys

PRIMARY KEY (column1, column2)

* + - FOREIGN KEY (constraints only enforced if using InnoDB storage)
      * Can only be coded at the table level (at the end of the CREATE TABLE statement)
        + CONSTRAINT constraintName

FOREIGN KEY (columnName) REFERENCES otherTableName (columnName)

* ON DELETE CASCADE
  + Including this code at the end of the constraint will delete all related records referenced (use with caution)
* ON DELETE SET NULL
  + Sets related records to NULL values (won’t use much or at all)
    - * Can set foreign keys at column level
        + REFERENCES otherTableName (columnName) as an attribute
    - AUTO\_INCREMENT
    - UNIQUE
    - NOT NULL
    - DEFAULT v (v is the value)
* Altering Tables
  + ALTER TABLE tableName *ACTION*
  + Actions
    - RENAME TO (will nename the table)
    - ADD columnName DATATYPE ATTRIBUTES (add a column using same format at CREATE TABLE)
      * AFTER columnName2 (will insert the added column after a specified column, use at end of ADD line after attributes)
    - DROP columnName (drop/delete a column and all of its data)
    - CHANGE (renames a column)
      * Must include the data type and all of the attributes for the column
      * ALTER TABLE tableName CHANGE origColName newColName DATATYPE ATTRIBUTES
    - MODIFY (changes definition of a column such as DATATYPE or ATTRIBUTES
      * Supply DATATYPE and all necessary attributes
      * Be careful about losing data if modifying DATATYPE
    - ALTER (set or drop default values)
      * …ALTER columnName SET DEFAULT v (where v is the value)
      * …ALTER columnName DROP DEFAULT (removes a default value)
* Indexes
  + Create an index
    - CREATE [UNIQUE] INDEX|KEY indexName ON tableName (columnName1 [ASC|DSC] [, columnName2 [ASC|DSC]…)
    - Can also do this when creating a table
      * At the end of defining columns and datatypes you can define
        + PRIMARY KEY (columnName1), UNIQUE INDEX indexName (columnName2), INDEX indexName2 (columnName3) etc.
  + Drop an index
    - DROP INDEX indexName ON tableName;
* Users/Priveleges
  + Creating Users
    - CREATE USER username@hostname IDENTIFIED BY ‘pa55word’;
      * If not @hostname specified, SQL uses a % which indicates the user can be from any host
      * Should specify the hostname in most cases, no hostname might mean that you can’t connect on some operating systems
  + DML privileges (most common for end users)
    - SELECT
    - INSERT
    - UPDATE
    - DELETE
  + DDL privileges (common for administrators/programmers)
    - CREATE
    - ALTER
    - DROP
    - INDEX
  + Other privileges
    - CREATE USER (can create, modify, and delete other users) (admins only)
    - ALL [PRIVILEGES] (grants all at the current level other than GRANT OPTION)
    - GRANT OPTION (Allos a user to grant his/her privileges to other users)
    - USAGE (doesn’t grant any privileges, can grant privileges later, doesn’t change existing privileges) (useful for changing passwords)
    - There are more (MySQL reference manual)
  + Privilege levels
    - \*.\* (all databases and all tables) (Global)
    - name\_db.\* (all tables on specified db) (Database)
    - name\_db.tableName (all columns on specified table) (Table)
    - (column1[, column2…]) name\_db.tableName (specified column in table in database) (Column)
  + Granting Privileges
    - GRANT PRIVILEGE1[, PRIVILEGE2]… ON privilegeLevel TO username@hostname [IDENTIFIED BY ‘pa55word’][, user2name [IDENTIFIED BY ‘pa55word2’]…] [WITH GRANT OPTION]
    - Possible to create/grant at once
      * add IDENTIFIED BY ‘pa55word’ after username@hostname
    - Possible to grant to multiple users at once
      * TO user1name@hostname [IDENTIFIED BY ‘pa55word1’], user2name@hostname [IDENTIFIED BY ‘pa55word2’]
    - Grant the GRANT OPTION
      * WITH GRANT OPTION; (at the very end of the grant statement)
    - Most typical version
      * GRANT SELECT, INSERT, UPDATE, DELETE ON databaseName\_db.\* TO username@hostname;
    - Create a database administrator
      * GRANT ALL ON \*.\* TO dba IDENTIFIED BY ‘supersecretpa55word’ WITH GRANT OPTION
      * Remember that ALL will only grant all of your privileges
    - Limit privileges to minimum necessary to avoid issues
  + Revoking Privileges (does not delete users)
    - REVOKE [ALL] [privilegeList], [GRANT OPTION] [ON [dbName.]table] FROM user1[, user2…]
    - Revoke all from an administrator level
      * REVOKE ALL, GRANT OPTION FROM username;
    - Revoke some stuff
      * REVOKE UPDATE, DELETE ON databasename\_db.tablename FROM user@hostname;
  + Viewing Privileges
    - View all users for the current server
      * SELECT User, Host from mysql.user; (exact syntax)
    - View privileges for a specific user
      * SHOW GRANTS [FOR username@hostname];
    - View privileges for the current user
      * SHOW GRANTS;
  + Renaming Users
    - RENAME USER oldname@hostname TO newname@hostname;
    - Privileges are transferred to the new name
  + Changing Passwords
    - GRANT USAGE ON \*.\* TO username@hostname IDENTIFIED BY ‘newpa55word’;
      * The grant usage on all database ensures that the password applies to all databases
      * Grant usage does not change the privileges that currently apply
  + Deleting Users
    - DROP USER username@hostname;

**Loading Data into a Database from a File**

* File Requirements
  + Must be a tab-delimited or comma separated values file
  + Must provide data for every column in the database
* phpMyAdmin method
  + Select the database
  + Select the table
  + Click the import tab
  + Select the file to import
  + Choose “CSV using LOAD DATA” (even for tab delimited txt files)
  + Need to set “field terminator” to \t (tab) if doing tab delimited txt file (otherwise use all defaults)
* Command prompt method
  + cd \xampp\mysql\bin

mysql -u root -p

Enter password: \*\*\*\*\*\*

use database\_name;

load data local infile “c:/directory/path/file.txt” into table tableName;

exit;

* MySQL Workbench method
  + Open the file and the table
  + Click on the inserts tab
  + Click on the “import records from an external file” button

**Dump a Database into an SQL Script (to recreate on a different server)**

* phpMyAdmin method
  + Select the database
  + Click the Export tab
  + Click “save output to a file” option (otherwise it just displays the script in the browser)
  + Can deselect the “Data” options if you just want the architecture and not the data
* Command Prompt method
  + cd \xampp\mysql\bin

mysqldump -u root -p databaseName > scriptFileName.sql

Enter password: \*\*\*\*\*\*

* + The saved file will be in the \xampp\mysql\bin directory

**Data Manipulation Language (DML)**

* Multiple SQL statements should be separated with a semicolon
  + The semicolon goes after the completion of the statement which could include multiple actions (such as SELECT, FROM, WHERE, & ORDER BY)
* SELECT
  + \* is the wildcard that will select every column from the table
  + FROM should be specified to say which table the data should come from
  + WHERE allows you to provide criteria to specify which rows (entries) are retrieved
    - Can use <, >, <=, >=, =, <> (not equal)
    - Numbers don’t need to be in quotes
    - Dates and strings do (WHERE date > ‘2007-03-10’)
    - Character comparisons are NOT case sensitive
    - Compound search criteria
      * AND, OR, NOT
      * Use parentheses for order of operations
      * List field name each time with criteria
    - NULL
      * Use IS NULL or IS NOT NULL for search criteria
    - LIKE
      * Use to specify a pattern to follow
      * % matches any string of 0 or more chars
      * \_ matches any single character
        + WHERE columnName LIKE ‘Fender%’ returns all rows that start with ‘Fender’
        + WHERE columnName LIKE ‘%cast%’ returns anything with ‘cast’ in it (can have other charactes both before and after the cast)
        + WHERE zip LIKE ‘294\_\_’ returns all zip codes that start with 294 as long as they are 5 digits
        + WHERE date LIKE ‘2007-06-\_\_%’ returns any date in June 2007, need % to include all times
  + ORDER BY lets you sort the data using ASC (default) for ascending or DESC for descending
    - Separate multiple sort parameters with commas
    - SELECT \* or SELECT column-1, column-2 … or SELECT first\_name, last\_name, address …

FROM tbl\_clients

WHERE subscr = TRUE

ORDER BY last\_name, acctBalance DESC

* + DISTINCT
    - SELECT DISTINCT will only return unique values (each value only once)
  + Use an alias to display a different column name
    - SELECT longColumnName AS name…
    - Use AS to show that it is an alias (AS is not required, but good practice to use)
    - Use “” to add spaces (SELECT productName AS “Product Name”)
    - Good idea to use aliases for calculated columns
  + LIMIT the number of rows returned
    - Add LIMIT to the end of the query
    - LIMIT [rowOffset, ] maxRows
      * skips the number in rowOffset and returns maxRows number of rows
  + Joins get data from related tables
    - Inner joins will only retrieve data if their related columns match (only retrieves sample codes in which the related table includes tl or age data)
    - JOIN specifies the second table
    - On specifies the column to be used for the join
    - Need to include table name for any columns that appear in both tables
      * columnName.tableName
      * Common to use table aliases (usually don’t code AS for table aliases)
      * Must use the alias in the code after it’s created
      * Aliases can be one or more letters
    - SELECT location, tl, age

FROM tbl\_sampling s

INNER JOIN tbl\_flounder f

ON s.sample\_code = f.sample\_code

WHERE species\_code = ‘A412’

ORDER BY date, tl

* + - Multiple tables can be joined, but do in order
      * s.\_\_ = o.\_\_ then o.\_\_ = i.\_\_ then i.\_\_ = r.\_\_
      * List all columns to include in the SELECT statement, then FROM, then all the joins, then WHERE, then ORDER BY
    - Outer joins include all of the rows from one of the tables even if there are not related data in the other table
      * Good at seeing all of one type and whether or not there are matching data in the other table
      * Also good for finding missing or empty data
      * LEFT OUTER
        + All rows from the first (left) table are included
      * RIGHT OUTER
        + All rows from the second (right) table are included
  + Summary Queries (aggregate functions)
    - Funtions & syntax (all but COUNT(\*) omit NULL values)
      * AVG(expression)
      * SUM(expression)
      * MIN(expression)
      * MAX(expression)
      * COUNT(expression) number of non-NULL values in the expression
      * COUNT(\*) total number of rows selected by the query
    - Provide aliases or the expression will be returned as the column name
    - Can string expressions into one query, each will return a row, separate with commas
    - SELECT MIN(columnName) AS minValue, MAX(columnName) AS maxValue, AVG(columnName) AS avgValue

FROM tableName

* + - GROUP BY
      * If you want the aggregate functions to work on separate values within a table (count of southernFlounder and sum of gulfFlounder over 355 mm tl)
        + SELECT sampler, speciesCode, COUNT(\*) AS numFlounder

FROM tbl\_sampling s INNER JOIN tbl\_flounder f

ON s.sampleCode = f.sampleCode

WHERE tl > 355

GROUP BY sampler, speciesCode

* + - * HAVING
        + Can be used after a GROUP BY clause and applies to each group (like where but where applies to each row and HAVING applies to each group)
        + Add HAVING COUNT(\*) > 1 would only include samplers that caught 2 or more of a particular species

HAVING is executed after the rows are grouped, WHERE selects data before grouping

* + - * + Having can use aggregate functions but WHERE cannot
  + Subqueries (SELECT within a SELECT statement)
    - Four ways to use
      * In a WHERE clause as a search condition
      * In a HAVING clause as a search condition
      * In the FROM clause as a table specification
      * In the SELECT clause as a column specification
    - WHERE
      * WHERE tl > (SELECT AVG(tl) FROM tbl\_flounder)
      * Selects records with total lengths larger than the average total length for all flounder
      * Depending on what the subquery returns (value, column, table) you can use it appropriately such as with IN columnName or FROM tableName
    - SELECT
      * SELECT speciesCode, speciesName, (SELECT COUNT(\*) FROM tbl\_flounder

WHERE tbl\_flounder.speciesCode = tbl\_species.speciesCode) AS numSpecies

FROM tbl\_species

* + - * WHERE [NOT] EXISTS (subquery)
        + SELECT s.sampleCode

FROM tbl\_samplers s

WHERE NOT EXISTS

(SELECT \* FROM tbl\_flounder f

WHERE s.sampleCode = f.sampleCode)

* + - * + Above code should return all sampleCodes that don’t have matches in the flounder table
  + UNION
    - Returns the combined results from two queries (does not repeat a row that is returned by both queries)
    - Syntax
      * the *columns* should be the same for both queries
      * columns and table are often:
        + V.name FROM Vendor V and C.name FROM Customer C
        + for columns with the same name in each table
      * SELECT (*columns*)

FROM *table1*

WHERE *conditions*

UNION

SELECT (*columns*)

FROM *table2*

WHERE *conditions*

ORDER BY *column*

* + INTERSECT
    - Returns only the rows that are selected by both queries
    - Sometimes INNER JOIN can produce the same results
    - Syntax
      * columns and table are often when column in both tables:
        + V.name FROM Vendor V and C.name FROM Customer C
      * SELECT (*columns*)

FROM *table1*

WHERE *conditions*

INTERSECT

SELECT (*columns*)

FROM *table2*

WHERE *conditions*

ORDER BY *column*

* + EXCEPT
    - Returns only the rows from the first select statement that aren’t found in the second select statement
    - Sometimes LEFT OUTER JOIN produces the same result
    - Syntax
      * columns and table are often when column in both tables:
        + V.name FROM Vendor V and C.name FROM Customer C
      * SELECT (*columns*)

FROM *table1*

WHERE *conditions*

EXCEPT

SELECT (*columns*)

FROM *table2*

WHERE *conditions*

ORDER BY *column*

* INSERT (action query)
  + Inserts a row into a table
  + Must specify the data to include in the entry
    - Can use the NOW() function for datetimes
    - Do not need to include data for automatically generated data (like an auto-incremented column, but you may need to specify the column list in this case)
    - Not required to supply data for columns that can be NULL, but must supply data for columns that cannot be NULL
    - Not required to supply data for columns that contain default values (but you can)
      * Use DEFAULT in the action query to specify default value
    - INSERT INTO table\_name [(column-list)] // [optional, but helpful]

VALUES (value\_list)

* + - If you don’t supply a column list, you must supply the VALUES in order that they appear in the table (probably good idea to supply the column list in many cases)
    - Can add multiple rows by placing comma after each set of values
    - INSERT INTO tbl\_flounder (sample\_code, coll\_date, specimen\_num, sex)

VALUES (180310, 3/10/2018, 1, ‘F’)

* UPDATE (action query)
  + Will update or change data in a table
  + Can use subqueries where necessary
  + UPDATE table\_name

SET columnName1 = expression-1 [, columnName2 = expression-2] …

WHERE selection\_criteria

* + Updating a column in one row
    - UPDATE products

SET productName = ‘Ludwig 5-Piece Kit with Zildjian Cymbals’ (string values in quotes)

WHERE productCode = ‘ludwig’ (only one Ludwig entry in products table)

* + Updating columns in multiple rows
    - UPDATE products

SET listPrice = 199

WHERE categoryID = 1 (every entry with categoryID of 1 will be updated)

* DELETE (action query)
  + Will delete one row from a table (single out its unique data like an auto-incremented number)
  + DELETE FROM table\_name

WHERE selection\_criteria

* + Deleting one row from a table
    - DELETE FROM table\_name

WHERE unique\_key = value (or some other unique identifier)

* + Deleting multiple rows from a table
    - DELETE FROM table\_name

WHERE field\_name > value (or other group identifier)

* + Deleting using a subquery
    - DELETER FROM tbl\_flounder

WHERE sampleCode IN

(SELECT sampleCode FROM tbl\_flounder

WHERE samplerName = ‘SCDNR’)

**phpMyAdmin**

* Access phpMyAdmin through the XAMPP control panel
  + Run the Apache and MySQL modules
  + Click on Admin button for MySQL
* Access phpMyAdmin through your browser
  + enter web address
  + localhost:8080/phpMyAdmin
* Names of databases will group them together if separated by underscores
  + my\_guitar\_shop1 and my\_guitar\_shop2 will be nested under phpMyAdmin trees under the directories my, guitar
* Privileges
  + root user has administrator privileges
  + To grant limited privileges to other users/create users
    - Execute SQL statements in SQL tab of phpMyAdmin
    - GRANT SELECT, INSERT, DELETE, UPDATE (as many or as few as you want etc.)

ON dataBaseName.\* (all tables in dataBaseName or dataBaseName.products 1 table only)

TO user\_name@localhost (their username)

IDENTIFIED BY ‘pa55word’ (their login password)

**Connecting to a Database (php code)**

* Creating a new object from a class
  + new *className(arguments)*;
  + Creating a database object from the PDO (php data object) class
    - new PDO(*$dsn, $username, $password*);
    - DSN syntax for a MySQL database
      * mysql:host=*host\_address*;dbname=*database\_name*
* Connecting to a MySQL database named ‘my\_guitar\_shop1’ located on the local host
  + Three arguments are required by MySQL ($dsn, $username, $password)
  + $dsn = ‘mysql:host=localhost:8080;dbname=my\_guitar\_shop1’; // specifies the db location

$username = ‘mgs\_user’; // assuming this username/password combo exists

$password = ‘pa55word’; // whatever password is assigned

$db = new PDO ($dsn, $username, $password); // creates PDO object

* Handling exceptions (when failing to connect to a database)
  + Use try/catch statements
    - The php interpreter will ‘try’ the first statement, and execute it fully if it succeeds
    - If any part of the try statement fails, then it immediately goes to the catch statement
    - Error messages are usually accessed and displayed in the catch statement
  + Handling PDO exceptions
    - try {

$db = new PDO($dns, $username, $password);

echo ‘<p>Your are connected to the database!</p>’;

} catch (PDOException $e) {

$error\_message = $e->getMessage();

echo “<p>An error occurred while connecting to the database: $error\_message </p>”;

}

* + Handling any type of exception
    - try {

// statements that might throw an exception

echo ‘<p>Whatever you want to say if it works, or skip this line</p>’;

} catch (Exception $e) {

$error\_message = $e->getMessage();

echo “<p>Error message: $error\_message </p>”;

}