Revision

Dynamic Web Development

SQL Essentials

A. STRUCTURING DATABASES WITH SQL & GUI

CREATE A DATABASE

Description: The CREATE DATABASE statement is used to create a new SQL database.

Syntax: **CREATE DATABASE** *dbname*;

In-Class Exemplar:

1. **Create our first database using SQL**

   CREATE DATABASE testDB;

   CREATE DATABASE studyDB;

2. **Create our second database using the phpMyAdmin GUI:**
   - Click on Home icon
   - Click on Databases
   - Enter new database name (studyDB2) & hit “Go”

NOTE: *Make sure to spell the queries correctly. An error will appear below if something is wrong and will try to provide a clue!*

DROP DATABASE

Description: Delete a database

Syntax: **DROP DATABASE** *dbname*;

In-Class Exemplar:

1. **Delete database using SQL**

   DROP DATABASE testDB;

2. **Delete database using the phpMyAdmin GUI (*needs to have active table*):**

- Click on Home & select the Databases tab
- Click the check box for the database to delete (testDB2) and click Drop

SELECT DATABASE

Description: We must always ensure we select the correct database or area in our phpMyAdmin

E.g. You may want to select a database to create a table in.

Syntax: USE *dbname*;

In-Class Exemplar:

1. **Select our studyDB to use queries on it**

```
USE studyDB;
```

CREATE TABLE

Description: The CREATE TABLE statement is used to create a new table in a database, like buckets, to allow data to be inserted into it in set fields.

The column parameters specify the names of the columns for the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

Syntax:

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

In-Class Exemplar:

1. **Create tblPersons, inside testDB, with 3 different fields including a primary key**

```
USE testDB;
CREATE TABLE tblPersons (
        PersonID int NOT NULL AUTO_INCREMENT,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Address varchar(255),
        City varchar(255),
        PRIMARY KEY (PersonID)
);
```

NOTE: *Auto_increment: always starts at 1 and increases by 1 for each field & IS NEVER RE-USED*

NOTE: *varchar: Stands for variable character & allows for storage of text-based characters*

2. **Create tables for studyDB database – Student Table**

```
USE studyDB;
CREATE TABLE tblStudent (
        StudentID int NOT NULL AUTO_INCREMENT,
        FName varchar(255) NOT NULL,
        LName varchar(255),
        Email varchar(255),
        PRIMARY KEY (StudentID)
);
```

3. **Create tables for studyDB database – Daily Study Time Table**

```
USE studyDB;
CREATE TABLE tblDailyTime (
        DailyTimeID int NOT NULL AUTO_INCREMENT,
        StudentID int NOT NULL,
        sDay DATE NOT NULL,
        NoHours int,
        PRIMARY KEY (DailyTimeID)
);
```

INSERT RECORD(S)

Description: The INSERT INTO statement is used to insert new records & data into a table.

Syntax:

```
INSERT  INTO table_name (column1,column2,column3,... ) VALUES (value1,value2,value3,...);
```

OR

```
INSERT INTO table_name VALUES (value1,value2,value3,...);
```

In-Class Exemplar:

1. **Insert Query for studyDB into Student Table**

   Let's now insert some data into our studyDB database. We will begin by adding some data to tblStudent table. NOTE: StudentID is AUTO INCREMENT so no need to state a value for this field.

```
USE studyDB;
INSERT INTO tblStudent (FName, LName, Email) VALUES ('Alex', 'Bicknell','alex@gmail.com');
```

2. **Insert Query for studyDB database into Student Table with Multiple rows**

   Let's now insert some data into our studyDB database. We will begin by adding some data to tblStudent table. NOTE: StudentID is AUTO INCREMENT so no need to state a value for this field.

```
USE studyDB;
```

INSERT INTO tblStudent (FName, LName, Email) **VALUES** ('Alex', 'Bicknell','alex@gmail.com'), ('Dan', 'Fitzsimmons', 'dan@gmail.com'), ('Ted, 'Davies, 'ted@gmail.com'), ('Mary', 'Thomas', 'mary@gmail.com');

3. **We can check these in the GUI to check they have been populated**

4. **Insert data for studyDB into Daily Study Time Table for One Time**

```
USE studyDB;
INSERT INTO tblDailyTime (StudentID, sDay, NoHours)
VALUES ('1', '2021-02-01', '3');
```

5. **Insert Query for studyDB database into Daily Study Time Table with Multiple rows**

```
USE studyDB;
INSERT INTO tblDailyTime (StudentID, sDay, NoHours)
VALUES ('1', '2021-02-05', '3'), ('1', '2021-02-20', '3'), ('1', '2021-02-30', '3');
```

**Show error & how to handle (as there is no February 30th)*

Add Data Visually using GUI

- Table Options -> Insert Tab

NOTE: We never insert a value for the PRIMARY KEY, as it does it by itself & auto-increment

SELECT RECORD(S)

The SELECT statement is used to select data from a database

Select ALL (*) from Student Table

```
USE studyDB;
SELECT * FROM tblStudent;
```

** Outputs all student rows under the tblStudent table*

Select ALL (*) from DailyTime Table

```
USE studyDB;
SELECT * FROM tbldailytime;
```

DELETE RECORD(S)

The DELETE statement is used to delete a row of data from a table

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

Syntax: **DELETE** **FROM** *table_name* **WHERE** *condition*;

Delete using the student's first name:

USE studyDB;

DELETE FROM tblStudent WHERE FName='Ted';

Delete using the student's ID:

USE studyDB;

DELETE FROM tblStudent WHERE studentID='3';

Using the GUI to delete a row:

- Select the table + click browse
- Click the delete option in the exact row

WHERE CLAUSE & BETWEEN ("*FILTERING*")

Description: The WHERE statement is used to *filter* our selected results

Syntax: SELECT *column_name,column_name* FROM *table_name* WHERE *column_name operator value*;

1. Select a specific student from studyDB

USE studyDB;

SELECT * **FROM** tblStudent **WHERE** FName= 'Mary' **AND** LName='Thomas';

2. Select specific student using just one field

USE studyDB;

SELECT * **FROM** tblStudent **WHERE** FName= 'Mary'

3. Select records based on a range (date) (*records where day is greater than 3 February 2021*)

USE studyDB;

SELECT * **from** tblDailyTime **WHERE sDay >** '2021-02-02';

4. Select records based on current date (*records where day is less than today's date*)

USE studyDB;

SELECT * **from** tblDailyTime **WHERE sDay <** CURDATE();

5. Select records between a certain date range

USE studyDB;

SELECT * **from** tblDailyTime **WHERE** sDay **BETWEEN** '2021-02-02' **AND** '2021-02-03';

UPDATE QUERY

The UPDATE statement is used to modify the existing records in a table.

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement.

The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

Used to change the value of columns….be sure to use WHERE otherwise all records will be changed!

UPDATE *table_name*
SET *column1=value1,column2=value2,…*
WHERE *some_column=some_value*;

UPDATE *tblstudent*
SET *LName='Brown'*
WHERE *studentID=3*;

UPDATE tblStudent SET LName='Brown' WHERE studentID=1;

GUI: Go to the table

- Click Browse & click the edit option for the specific row
- Can update/edit values in that row
- NOTE: Whenever we use the GUI, it displays the SQL query

SETTING UP FOREIGN KEY

So far, we have created foreign key fields, but have not linked them to the actual keys from which their value derives

Create new database with foreign key

- **Setup new database**

  CREATE DATABASE studyDB2;

- **Setup new tables (tblStudent) & insert some fake data into the table (x2 rows)**

```
CREATE TABLE tblStudent(
        StudentID int NOT NULL AUTO_INCREMENET,
        FName varchar(255) NOT NULL,
        LName varchar(255),
        Email varchar(255),
        PRIMARY KEY (StudentID)
);

INSERT INTO tblStudent (FName, LName, Email) VALUES ('Alex', 'Bicknell', 'alex@gmail.com'),
('Dan', 'Fitzsimmons', 'dan@gmail.com');
```

- **Setup second table which refers to foreign key (foreign key at bottom which REFERENCES where it comes from**

  CREATE TABLE tblDailyTime (

  DailyTimeID int NOT NULL AUTO_INCREMENT,

  StudentID int NOT NULL,

  sDay DATE NOT NULL,

  NoHours int,

  PRIMARY KEY (DailyTimeID),

  FOREIGN KEY (Student ID) REFERENCES tblStudent(StudentID)

  );

- **The GUI under the new table & click on Insert - if we select the foreign key, it displays a drop down menu (showing us its linked)**

- **Lets now add data into our 2nd dailytime table & check our foreign key is working**

  INSERT INTO tblDailyTime (StudentID, sDay, NoHours) VALUES ('1', '2021-02-01', '3')

  Can see that the values have been added correctly

(2) Updating existing database to provide for foreign key (could drop table BUT not ideal)

- Select the database & table we wish to add foreign key to (studyDB -> tblDailyTime)
- Click on Structure tab → Relation view
- Add new column **OR** add existing column (we select existing StudentID)
- Specify the database & table & column it is going to link to (INNODB) and save
- NOTE the SQL used to adjust this database:

  ALTER TABLE tblDailyTime ADD FOREIGN KEY (StudentID) REFERENCES tblStudent (StudentID) ON DELETE RESTRICT ON UPDATE RESTRICT;

- Check our GUI in our table - can see our foreign key is now linked!

TESTING QUERIES

SEE POWERPOINT FIRST

Simple way to test queries (that totals the total number of student hours studied)

SELECT SUM(NoHours) FROM tblDailyTime;

Expected Result from Table GUI: 16

Actual Result: 16

Correct: Yes!

Testing:

SELECT MIN(NoHours) As MinimumHours from tblDailyTime WHERE sDay BETWEEN '2021-02-02' AND '2021-02-28';

Expected Result: 3

Actual Result: 3

Correct: Yes

Testing:

SELECT MAX(NoHours) As MinimumHours from tblDailyTime WHERE sDay BETWEEN '2016-02-02' AND '2016-02-03';

Expected Result: 4

Actual Result: 4

Correct: Yes

EXPORT & IMPORT DATABASE FILES

Export:

- Select database & click export tab
- Drop down menu and select the appropriate export format (in our case ¯ "SQL") & hit Go
- The client will generate the SQL needed to generate the database. Copy the text & paste into a plain text file (NotePad & save to hard drive)

Import:

- Create new database (use studyDB / dropped db)
- Select database and click on SQL tab
- Paste the SQL saved into the SQL tab. Hit Go.
- Database will be generated all through a series of SQL commands AND tables repopulated with data exported!

B. SQL COMMANDS AGAINST RECORDS

BASIC SQL COMMANDS

Description: Functions that help to pool or sort data according to function

1. SUM() – returns total value of numeric column

Syntax: SELECT SUM(column_name) FROM table_name;

EXAMPLE: Total No. of Hours studied in entire table:

USE studyDB;

SELECT SUM(NoHours) FROM tblDailyTime;

GUI: To double check, click on the table, click on Browse & manually count the hours in the column

EXAMPLE: Total noHours studied between 2021-02-02 and 2021-02-03

USE studyDB;

SELECT SUM(NoHours) As TotalHours FROM tblDailyTime WHERE sDay BETWEEN '2021-02-02' AND '2021-02-28';

- "As": TotalHours creates a variable which stores the result of the SUM

EXAMPLE: Variation (no variable of TotalHours)

USE studyDB;

SELECT SUM(NoHours) FROM tblDailyTime WHERE sDay BETWEEN '2021-02-02' AND '2021-02-28';

2. AVG() - returns the average value of a numeric column.

Syntax: SELECT AVG(column_name) FROM table_name

EXAMPLE: Average noHours studied between 2021-02-02 and 2021-02-28

USE studyDB;

SELECT AVG(NoHours) As AverageHours FROM tblDailyTime WHERE sDay BETWEEN '2021-02-02' AND '2021-02-28';

3. MAX() - returns the largest value of the selected column.

Syntax: SELECT MAX(column_name) FROM table_name;

EXAMPLE:

SELECT MAX(NoHours) As MaximumHours from tblDailyTime WHERE sDay BETWEEN '2021-02-02' AND '2021-02-28';

4. MIN() - returns the smallest value of the selected column.

Syntax: SELECT MIN(column_name) FROM table_name;

SELECT MIN(NoHours) As MinimumHours from tblDailyTime WHERE sDay BETWEEN '2021-02-02' AND '2021-02-28';

SPECIFIC SQL COMMANDS

1. COUNT() – returns the *number* of rows that matches a specified criterion

Syntax: SELECT COUNT(*column_name*) FROM *table_name* WHERE *condition*;

NOTE: The WHERE condition is optional – can just count ALL rows

EXAMPLE: Count the Number of Time Entries there are for StudentID 1 in entire table:

USE studyDB;

SELECT COUNT(NoHours) FROM tblDailyTime WHERE StudentID=1;

NOTE: COUNT(*) – returns the number of rows in a specified table, and it preserves duplicate rows

2. GROUP BY – groups rows that have the same values into summary rows, like "find the number of customers in each country".

Syntax: SELECT *column_name(s)* FROM *table_name* WHERE *condition*
GROUP BY *column_name(s)*

NOTE: The GROUP BY statement is often used with aggregate functions
(COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

EXAMPLE: List the number of student entries made for each different study day in February:

USE studyDB; SELECT COUNT(StudentID), sDay FROM tblDailyTime

GROUP BY sDay;

3. ORDER BY - used to sort the result-set in ascending or descending order (ascending by default, state DESC if you want descending)

Syntax: SELECT *column_name(s)* FROM *table_name* WHERE *condition*
GROUP BY *column_name(s)*
ORDER BY *column_name(s);*

EXAMPLE: List the number of student entries made for each different study day in February, in order of date:

USE studyDB; SELECT COUNT(StudentID), sDay FROM tblDailyTime

GROUP BY sDay

ORDER BY sDay;

4. HAVING - added to SQL because the WHERE keyword cannot be used with aggregate functions

Syntax: SELECT *column_name(s)* FROM *table_name* WHERE *condition*
GROUP BY *column_name(s)*
HAVING *condition*
ORDER BY *column_name(s);*

EXAMPLE: List the number of student entries made for each different study day in February, having StudentID greater than 3

USE studyDB; SELECT COUNT(StudentID), sDay FROM tblDailyTime

GROUP BY sDay

HAVING(StudentID) > 3;

SQL DATA TYPES

Broad Documentation: https://www.w3schools.com/sql/sql_datatypes.asp

1. STRING – Data types that store anything using alphabet text

Core subtypes: CHAR(size), VARCHAR(size), TEXT(size)

- **CHAR**: a fixed length string data type, so any remaining space in the field is padded with blanks (max 8,000 characters).
- **VARCHAR**: a variable length string data type, so it holds only the characters you assign to it (max 8,000 characters).
- **TEXT**: a variable length datatype, with a maximum of 65,000 characters / 2GB of data.

GENERAL USE NOTE: *If something has a fixed length, we **use CHAR**. If it has variable length with a well defined upper limit then **use VARCHAR**. If it's a big chunk of text that you have little control over **then TEXT** would be probably your best bet.*

2. NUMERIC – Data types that store value via numeric digits

Core subtypes: INT, BOOLEAN/TINYINT(1)

- **INT:** A medium integer. Signed range is from -8388608 to 8388607.
- **BOOLEAN/TINYINT(1):** Both are synonyms. Using binary, 0 designates false and 1 (or any other number) designates true.

3. DATE & TIMES – Date & time object which is in a set pattern and calculated

Core subtypes: DATE, TIME

- **DATE:** Store a date only. From January 1, 0001 to December 31, 9999
- **TIME:** Store a time only to an accuracy of 100 nanoseconds

NOTE: *MySQL 5.5 Manual - Date and Time Functions - https://dev.mysql.com/doc/refman/5.5/en/date-and-time-functions.html*

SQL SYNTAX

1. LIKE – used in a WHERE clause to search for a specified pattern in a column.

NOTE: There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

Syntax: SELECT *column1, column2, ...* FROM *table_name* WHERE *column_name* LIKE *pattern*;

EXAMPLE: Select all students with a first name starting with "a"

```
USE studyDB;
SELECT * FROM tblStudent WHERE FName LIKE 'a%';
```

EXAMPLE: Select all students with a last name ending with "s"

```
USE studyDB;
SELECT * FROM tblStudent WHERE LName LIKE '%s';
```

EXAMPLE: Select all students with a name with "a" in any position

```
USE studyDB;
SELECT * FROM tblStudent WHERE FName LIKE '%a%';
```

EXAMPLE: Select all students with a name that has "a" in the second position

```
USE studyDB;
SELECT * FROM tblStudent WHERE FName LIKE '_a%';
```

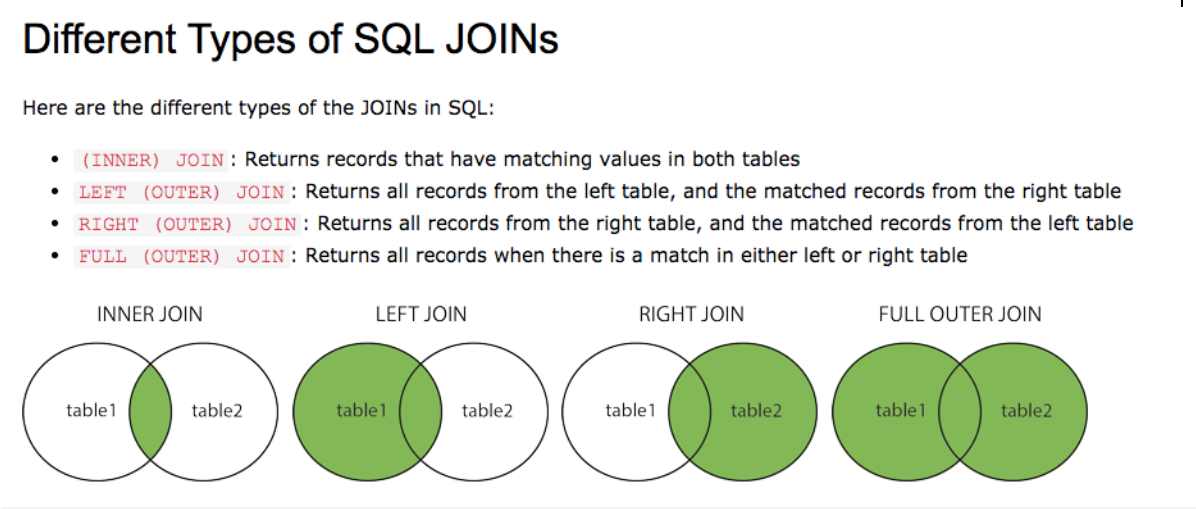
2. SELECT DISTINCT – return only distinct (different) values.

NOTE: Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values. REMOVES DUPLICATE RECORDS.

Syntax: SELECT DISTINCT *column1*, *column2, ...* FROM *table_name*;

SQL JOINS

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.



The most common type of join is: **SQL INNER JOIN ("simple join")**. An SQL INNER JOIN returns all rows from multiple tables where the join condition is met."

Source: http://www.w3schools.com/sql/sql_join.asp

```
SELECT tblDailyTime.DailyTimeID, tblDailyTime.StudentID, tblDailyTime.sDay, tblDailyTime.NoHours,
tblStudent.FName, tblStudent.LName, tblStudent.Email
FROM tblDailyTime
INNER JOIN tblStudent
ON tblDailyTime.StudentID = tblStudent.StudentID;
```

NOTE: *This is pretty complex. It will be unlikely we use this – but its worth looking into. Basically, its done to "select" mentions of an ID across the database to see where they are mentioned and how they could be joined.*