Databases Lab 1 - Introduction to SQL

Objectives:

• To introduce some basic SQL commands

• To provide practice at using simple SELECT commands to extract data

• To obtain experience at constructing simple SELECT commands

Background Knowledge:

This worksheet does not assume any prior experience. The main tool used for this course is an online IDE provided by Repl.it (<https://repl.it>). You are required to create a new account on Repl.it. Please make sure to not share links to your account or code on any forum (e.g. Moodle, teams) or with any of your classmate.

Creating an account on Repl.it:

Repl.it is a hosted development environment that provides workspaces for developers to practice coding and also deploy to a web server. It provides support for many programming languages such as PHP, Python, C, Java and SQLite. You can create an account by using your email, Google account or GitHub account. The repls dashboard is a place to manage and keep track of all your repls (workspaces) and includes the version control software.

To create a new repl (workspace), click on the "new repl" button on the top right-hand corner of your screen and then search for “SQLite”. After selecting SQLite, give your project a name and click on the create repl button. This would create a blank project consisting of a single file (main.sql). To begin, type the command ".help" in the main.sql file and click on the run button (a green button located at the top of your screen). This would run the command written in the main.sql file and present it on a terminal at the right-hand side of your screen.

Running the ".help" command shows all the common commands.

Here are two good resources to learn SQLite basic commands –

<https://sqlite.org/cli.html>

<https://www.sqlitetutorial.net/>

SQLite and other databases:

SQLite is an embedded relational database management system (RDBMS) and is serverless. There is no separate server process for managing the databases. It is different from some of the popular Client-Server RDBMS (MySQL, SQL Server, Oracle and so on). Client-Server DBMS is able to manage data between a system to others as a shared repository of enterprise data while SQLite can manage data as local data for individual applications and devices.

Creating a table and adding data:

You will have to set up a new table and add data to it yourself. The name of the table should be "pet". Create the table using the following command:

CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20), species VARCHAR(20), sex CHAR(1), checkups SMALLINT UNSIGNED, birth DATE, death DATE);

To check the table that you have just created, use this command:

“. schema” or “. schema table\_name”

To check the list of tables, use this command:

“. table”

To enter data, use the following command:

INSERT INTO pet (name,owner,species,sex,checkups,birth,death)VALUES

('Fluffy','Harold','cat','f',5,'2001-02-04',NULL),

('Claws','Gwen','cat','m',2,'2000-03-17',NULL),

('Buffy','Harold','dog','f',7,'1999-05-13',NULL),

('Fang','Benny','dog','m',4,'2000-08-27',NULL),

('Bowser','Diane','dog','m',8,'1998-08-31','2001-07-29'),

('Chirpy','Gwen','bird','f',0,'2002-09-11',NULL),

('Whistler','Gwen','bird','',1,'2001-12-09',NULL),

('Slim','Benny','snake','m',5,'2001-04-29',NULL);

Selecting Data from a Database:

The select command is the key command that is used to query data within a database and obtain results.

The simplest form is:

SELECT \* FROM table;

which simply returns all data from all columns within the table. Try this command on the pet table by replacing “table” in the above query by “pet” and see what data is in the pet table. Compare the data values in each column with the data types you identified in the previous step - are they as you would have expected?

In general, when we do not wish to select all data from a table, we can be more selective by adding the WHERE clause to the select statement. What do you think the following command does? Try it and see if you were correct.

SELECT \* FROM pet WHERE sex = "m";

We can get even more specific by replacing the \* in the SELECT statement with the names of the columns which we would like to retrieve. Try changing the previous SELECT statement to return the names of the owners of male animals.

If you run SELECT owner FROM pet; you should notice that the list contains one of the owner names more than once. Why do you think this is? You can remove duplicate values by using the qualifier DISTINCT immediately after the keyword SELECT - try it and see what happens.

If you want to obtain more than one column, you can simply supply a list of column names separated by commas. Similarly, if you want to select using more than one criteria, you can connect logical statements in the WHERE clause using the standard Boolean connectives AND, OR and NOT. For example, try the following,

SELECT name, species, sex FROM pet WHERE species = "snake" OR species = "bird";

Now try to formulate queries for the following, and then try them out using the SQLite client. Note that some of these problems do not ask you to return all the fields that you are selecting on, and so it can be difficult to work out whether you have got the correct results. You may need to initially run the queries as SELECT \*, and when you are sure you are getting the correct records, supply the list of column names you want.

Q1-1. The names of owners and their pet's name for all pets who are female.

Q1-2. The names and birth dates of pets which are dogs.

Q1-3. The names of the owners of birds.

Q1-4. The species of pets who are female.

Q1-5. The names and birth dates of pets which are cats or birds.

Q1-6. The names and species of pets which are cats or birds and which are female.

Comparison Operators and Ordering Output:

In addition to straight equality tests, you can use other operators to test for ordering. For example, try the following:

SELECT name FROM pet WHERE sex < "m";

The above example is slightly unusual. It would be normal to use tests for order on numeric data rather than string data. However, the example demonstrates that ">", ">=", "<=" and "<>" (for "not equal to") can be used on numeric and string data, as you might have expected. Now, write a select statement which will return all pet names which start with any letter at or above "F" in the alphabet. What happens if you rewrite the query replacing "F" with "f"? Now try to find all pets whose names start with "F" or above and whose owner's name also starts with "F" or above.

Q2-1. The names of owners and their pets where the pet's name ends with “er” or “all”

Q2-2. The names of any pets whose owner's name contains an "e"

Q2-3. The names of all pets whose name does not end with "fy"

Q2-4. All pet names whose owners name is only four characters long

Q2-5. All owners whose names begin and end with one of the first five letters of the alphabet

Q2-6. Repeat the previous query, but make the query sensitive to the case of letters of the alphabet the characters in the name *(NOTE: This query is not part of the assessed coursework!)*

You will have noticed that the results of many of your queries have, seemingly, been unordered. In fact, the results are produced in the order in which they are encountered when working through the table of data from top to bottom. Commands are provided to allow you to order the data returned to you. Try the following queries:

SELECT name, birth FROM pet ORDER BY birth;

SELECT name, birth FROM pet ORDER BY birth DESC;

SELECT name, species, birth FROM pet ORDER BY species, birth DESC;

These should be fairly obvious queries, but note that the order of column names is important in the last example. Try switching the position of species and birth in the last example to see what happens. Also, note that DESC only applies to the column immediately preceding it. Try moving DESC to just after species in the final query and see what happens.

Date Manipulation:

Dates occur commonly within data but are difficult to deal with because of the many formats in which dates can appear. To overcome this difficulty, most DBMS store dates in a single format internally, and provide functionality that the user can access to compare and manipulate date values. They also often provide a fixed number of date representations that can be used for the input and display of valid dates. This section will only look at a few of these facilities, and you should be aware that different DBMS will provide different methods for dealing with dates. Try the following:

SELECT name FROM pet WHERE strftime('%m',birth) = strftime('%m','now');

What do you think the query is seeking to do? If you do not get any names output, try again with the next month or the previous until you get some output.

You can also use functions to produce new columns in the result which are based on calculations performed on values within each record. For example, try the following:

SELECT name, strftime('%m', birth) FROM pet;

If you are observant you will notice that the title of the second column is now "*STRFTIME()*". This may be what we want, but when including a calculated value, we will more often want a descriptive column name rather than the calculation. We can achieve this by using an *alias*, as illustrated in the following query. Try it and look at the resulting column headings.

SELECT name, strftime('%m', birth) AS month FROM pet;

If you look carefully at the results you got, you will find that you also calculated the age of the animal which has already died. You can exclude this animal from the calculation of age by the clause WHERE death IS NULL. You will notice that for most of the animals the death field is reported as NULL. This means that no value has yet been supplied for this field. In general, we will try to avoid the use of fields which can be NULL, but in places where it is unavoidable you can test for a NULL value using IS NULL and for the converse using IS NOT NULL.

Counts and Value Manipulation:

As well as manipulating dates, we can use functions to manipulate values. Consider, for example, the following query:

SELECT owner, name, (checkups \* 20) AS income FROM pet;

This query uses the visits field to calculate how much money has been spent on each pet due to visits to the vet. There are many such numeric functions available, listed in the Numeric Functions section of the SQLite Manual. As you should expect by now, they can also be applied in the WHERE clause of the query. Perhaps more interesting, however, are the so-called "grouping" functions. Consider the following query:

SELECT owner, name, birth, MIN(strftime('%Y',birth)) AS birth FROM pet GROUP BY owner;

Enter this query and look at the output. What do you think this query is doing, and what is the relevance of the GROUP BY clause?

Now try to formulate a query to find the following (Note: look at the Counting Rows and Group By Functions sections of the SQLite Manual for information which will help you to create these queries):

Q3-1. The average number of check-ups that each owner has made with their pets

Q3-2. The number of pets of each species in ascending order

Q3-3. The number of pets of each species that each owner has

Q3-4. The number of distinct species of pet each owner has

Q3-5. The number of pets of each gender there are in the database, where the gender is known

Q3-6. The number of birds each owner has

Q3-7. The total number of check-ups each owner has made with all their pets

MSc Coursework:

This lab constitutes for 5% of your overall unit.

Submit 18 SQL queries, from those three sets of queries on the sheet (sets labelled Q1, Q2, Q3).

Each query will be marked as either correct or incorrect. A correct query will give you 4 marks for each query in set Q1, 5 marks for each query in set Q2 and 6 marks for each query in set Q3.

Queries are worth a total of 91 marks (= 6x4 + 5x5 + 7x6). 9 marks are discretionarily reserved for the correct formatting and presentation of lab submission. Poor presentation and failing to adhere to the submission instructions (e.g., Wrong file type, wrong naming of files) may result in a penalty.

How to submit:

The coursework must be submitted via Moodle. All SQL queries along with the repl.it link to your code should be collected in a single .txt file with the filename YourBathUsername\_Lab\_1.txt. Make sure your repl.it file is not edited after the submission date and the code matches the one that is submitted in the .txt file. The labs are not marked anonymously.

Feedback:

Written individual feedback will be provided through Moodle within 3 weeks of the

submission deadline. Clarification of this feedback can always be obtained for the unit

lecturer.

You are encouraged to evaluate your answers before submission. All queries can be easily tested manually. If in doubt, ask one of the tutors in the lab.

Plagiarism:

As this is individual coursework and you need to be mindful about collusion. Collusion

which is a form of plagiarism where two or more people work together to produce a piece of

work all or part of which is then submitted by each of them as their own individual work.

Plagiarism is a serious academic offence but more often than not it results from

misunderstanding rather than a deliberate intention to cheat. Many students simply do not

understand what plagiarism is. Working The University of Bath and the Department of

Computer Science takes this offence very seriously. On additional information of what may be

considered plagiarism and different types of it, refer to the relevant Library resource.

(<http://www.bath.ac.uk/library/help/infoguides/plagiarism.html> )

Extension requests:

Please note that any requests for extensions to coursework deadlines should be submitted to the Director of Studies (Dr Marina De Vos). Note that you will need to submit a QA16 form to Marina to ask for an extension.