A Brief Introduction to SQL

SQL stands for Structured Query Language.. It is the standard language for relational databases. Basically every relational DBMS (RDBMS) implements SQL, although with slight variations.

In this section we will provide a brief introduction to SQL in the context of a simple example. We will expand on all of the material later.

Imagine we want to store a list of students enrolled in a class in a database, so we can later produce reports on that data. We will need to perform three kinds of actions, in roughly the following order:

1. Create the table where we will store the data. The SQL command we will use is **CREATE TABLE**
2. Put the data in the database; for this the first SQL command would be **INSERT**, which inserts a new row in a table. Later, we may want to execute **DELETE** commands to delete some rows, and **UPDATE** commands to change the values of some fields for some rows.
3. Get the data out for different reports, and to verify we entered the correct data. For this we use the **SELECT** command.

A note about syntax

SQL is not case sensitive, you can write SQL commands in uppercase, lowercase or any combination; however, it is customary to write reserved words in UPPERCASE to clearly distinguish them from names of fields and tables, that will vary. I follow this convention to make it easier for you to remember what is part of SQL syntax and what varies.

SQL also ignores extra spaces and the number of lines a statement uses. I will format the SQL statements for readability, but they can be entered in one line if necessary. Also, most text-based SQL clients use a semicolon (;) to terminate a statement, although the semicolon is not officially part of the SQL statement.

Another important issue is that SQL uses single quotes for character data, instead of double quotes, so the string constants are written like 'Orlando' instead of “Orlando”.

Finally, remember that SQL was designed to resemble English, but it is still a programming language. It uses less special symbols and more reserved words than most other programming languages. In particular, the boolean operators are the actual words AND, OR and NOT rather than special symbols.

Creating the table

OK, so we want to create the table. The fields we want to keep track of are:

- The student's Id, which we will call Id. This is a number, with 5 digits. This is the primary key.
- The student's name (just one field, rather than first/middle/last), which is a string. For SQL, we need to specify the maximum length; to simplify lets set it at 20.
- The student's Major, which is a string of three characters.
- The student's Age, which is a number with two digits.

So, our CREATE TABLE command would look like this:

```
CREATE TABLE Student (
    Id    NUMERIC(5) PRIMARY KEY,
    Name  VARCHAR(20),
    Major CHAR(3),
    Age   NUMERIC(2)
);
```

Example 1: CREATE TABLE statement

Adding Data

To add a new row to the table, we use the INSERT INTO command. We put the name of the table, a list of fields, the keyword VALUES, and a list of values. Remember that character or string constants go between single quotes.

So, the SQL command for inserting a row would look like this:

```
INSERT INTO Student(Id,Name,Major,Age) VALUES(1,'Orlando Karam','CS',21);
```

Retrieving Data

We retrieve data from a table with the SELECT command. We write the SELECT keyword, then a list of the fields we want (or * for retrieving all fields), and then the FROM keyword and the table we want to get data from. Normally we don't want to get data from all rows, so we use the WHERE keyword to specify conditions that the retrieved rows must satisfy.

The simplest SELECT statement retrieves all data in a given table, as shown in Example 2.

```
SELECT *
FROM Student
```

Example 2: Simplest SELECT statement, retrieves all fields and all rows from the Student table

If we don't want to get all fields, we use a list of the fields, separated by commas. We can also specify a condition on the WHERE clause. For example, if we want the Id and Name of all students majoring in 'CS' we would write:

```
SELECT Id,Name
FROM Student
WHERE Major='CS'
```

Example 3: Retrieving Id and Name of students majoring in 'CS'

We can of course combine conditions in the WHERE clause using AND or OR (notice that in SQL we use the actual words AND, OR, rather than special symbols), and can use the keyword NOT to invert the condition.

So, if we wanted to get the Id, Name and Age of all students who are not majoring in 'CS' and who are

older than 20 we would write:

```
SELECT Id,Name,Age
FROM Student
WHERE Major!='CS' AND Age>20
```

Example 4: Retrieving Id , Name and Age of students older than 20 and not majoring in 'CS'

Deleting Data

If we want to delete some rows, we can use the DELETE statement. After the DELETE keyword we write the table we are deleting from, and, normally, a WHERE clause specifying which rows to delete (we delete all rows if we don't put a WHERE clause).

For example, to delete the student with Id 1 we would write:

```
DELETE
FROM Student
WHERE Id=1
```

Example 5: Deleting the row for the student with id 1

Changing data

To change the data in some rows, we use the UPDATE statement. After the UPDATE keyword we write the name of the table, and then the SET keyword, and then a list of assignments, field=value; we normally specify a WHERE clause to limit the changes to only certain rows. For example, to change the row with Id 1 so its name is Alfredo Karam and its age is 25, we would write:

```
UPDATE Student
SET Name='Alfredo Karam', Age=25
WHERE Id=1
```

Example 6: Changing name and age for the student with id 1

What more is there

We have covered the basics of SQL, enough to let you start creating simple tables, but we've just scratched its surface. We will later learn how to add more constraints to the CREATE TABLE statements, so we can guarantee the data entered is not obviously wrong. We will also see how to eliminate (drop) the tables, and how to create indices to speed up retrieval. We will also learn all the ins and outs of the SELECT statement, allowing us to express complex conditions and to combine the information in several tables. In terms of adding data, we will see how to define transactions so we can update several tables at the same time.