7 Relational databases and structured query language (SQL)

Learning objectives:

- *Be able to use SQL to retrieve data from a relational database, using the commands*
  - *SELECT*
  - *FROM*
  - *WHERE*
  - *ORDER BY...ASC | DESC*
- *Be able to use SQL to insert data into a relational database using the command:*

  INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...)
- *Be able to use SQL to edit and delete data in a relational database using the commands:*

  UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition

  DELETE FROM table_name WHERE condition

7.2 Structured Query Language

Querying a database

The main purpose of storing data in a database is to enable applications to interrogate the database for information. This interrogation is called **querying the database**.

Structured Query Language (SQL)

Structured Query Language (SQL) can be used to query a database. It is a simplified programming language.

Retrieving data from a single table

Table 7.2.1 shows data for the **Student** table with structure

Student (StudentId, StudentName, Gender)

The following query, expressed in SQL, will retrieve all of the data in the **Student** table

```
SELECT *
  FROM Student;
```

The wildcard character * matches the attribute/field list

StudentId, StudentName, Gender

StudentId	Student Name	Gender
1	Ames	M
2	Baloch	F
3	Cheng	F
4	Dodds	M
5	Groos	M
6	Smith	F

Table 7.2.1 Table Student

The ANSI/ISO SQL standard requires that a semicolon is used at the end of the SQL statement but some systems relax this requirement. When writing SQL the convention is to use upper case for the SQL commands.

If we wanted just the data for *StudentName* we would refine the query as follows

```
SELECT StudentName
  FROM Student;
```

We could refine the search even further by adding a WHERE clause that applies a search condition as follows

```
SELECT StudentName
  FROM Student
    WHERE Gender = 'F';
```

The result set that would be returned when this query is applied to table **Student** would be as follows

Baloch
Cheng
Smith

because only these rows of the table match the search condition Gender = 'F'.

Gender = 'F' is actually called a predicate because it evaluates to either TRUE or FALSE.

If we also wanted the values of *StudentId* returned then the query would be

```
SELECT StudentId, StudentName
  FROM Student
    WHERE Gender = 'F';
```


Questions

1. Write an SQL query that returns the names of all students in *Table 7.2.1* who are male.

Retrieving data from multiple tables

Table 7.2.2 shows data in table form for the **Ward** table with structure

Ward (WardName, NurseInCharge, NoOfBeds)

WardName	NurseInCharge	NoOfBeds
Victoria	Sister Bunn	30
Aylesbury	Sister Moon	40

Table 7.2.2 Table Ward

Table 7.2.3 shows data in table form for the **Patient** table with structure

Patient (PatientId, Surname, *WardName*)

PatientId	Surname	WardName
1	Bond	Aylesbury
2	Smith	Victoria
3	Jones	Aylesbury
4	Biggs	Victoria

Table 7.2.3 Table Patient

The two tables are linked via a shared or common attribute *WardName*. The existence of an attribute common to both tables is not enough to join data from the corresponding tables correctly, as the following SQL query demonstrates

```
SELECT Ward.WardName, Ward.NurseInCharge,
       Patient.PatientId
  FROM Ward, Patient;
```

The part of the query `Ward.WardName` references the *WardName* attribute in table **Ward** and the part `Patient.PatientId` references *PatientId* attribute in table **Patient**.

The `FROM Ward, Patient` part joins both relations without regard for the way that the data is actually linked via matching values of the shared attribute, *WardName*. The result set returned by the query is shown in *Table 7.2.4*.

Victoria	Sister Bunn	1
Victoria	Sister Bunn	2
Victoria	Sister Bunn	3
Victoria	Sister Bunn	4
Aylesbury	Sister Moon	1
Aylesbury	Sister Moon	2
Aylesbury	Sister Moon	3
Aylesbury	Sister Moon	4

Table 7.2.4 Result set ignoring relationship between Ward and Patient

When the search condition

```
WHERE Ward.WardName = Patient.WardName
```

is added to the SQL query, we are able to exclude values that are not linked by the attribute *WardName* and to include only those that are. This SQL query will return the result set that corresponds to the real world situation shown in *Table 7.2.5*.

```
SELECT Ward.WardName, Ward.NurseInCharge, Patient.PatientId
  FROM Ward, Patient
    WHERE Ward.WardName = Patient.WardName;
```

Aylesbury	Sister Moon	1
Victoria	Sister Bunn	2
Aylesbury	Sister Moon	3
Victoria	Sister Bunn	4

Table 7.2.5 Result set taking account of relationship between Ward and Patient

The two relations have been joined on their common attribute, *WardName*, i.e. where the value of *WardName* is the same in both tables.

Writing the query as follows would return the same result set because dropping the table name prefix before *NurseInCharge* and *PatientId* in the SELECT part of the SQL query is allowed where there is no ambiguity as to what is intended.

```
SELECT Ward.WardName , NurseInCharge, PatientId
  FROM Ward, Patient
    WHERE Ward.WardName = Patient.WardName;
```


Questions

2 Write the SQL query that returns from *Tables 7.2.2* and *7.2.3* the name of the nurse in charge of the ward, surnames of all patients in this ward and the ward name.

Ordering the result set returned by a query

We can order a result set returned by a query in ascending or descending order with the keyword ORDER BY qualified by one of the keywords ASC or DESC. If the qualifier is omitted then ASC is assumed. For example, we can place the result set returned in ascending order on *WardName* by the query opposite.

```
SELECT Ward.WardName, NurseInCharge, PatientId
  FROM Ward, Patient
    WHERE Ward.WardName = Patient.WardName
      ORDER BY Ward.WardName ASC;
```

Table 7.2.6 shows the outcome of applying this query to the **Ward** and **Patient** tables.

Questions

3 Write the SQL query that returns the names of both nurses and their patients, from *Tables 7.2.2* and *7.2.3*, ordered in descending patient name order.

Aylesbury	Sister Moon	1
Aylesbury	Sister Moon	3
Victoria	Sister Bunn	2
Victoria	Sister Bunn	4

Table 7.2.6 Result set ordered on WardName in ascending alphabetic order

Relational or comparison operators for search condition

Table 7.2.7 shows comparison operators that may be used in SQL queries.

Comparison Operator	Description
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

Table 7.2.7 Comparison operators for SQL queries

Table 7.2.8 shows the outcome of applying this query to the **Patient** table.

```
SELECT PatientId, Surname
  FROM Patient
    WHERE PatientId <> 2;
```

1	Bond
3	Jones
4	Biggs

Table 7.2.8 Result set for PatientId <> 2

Table **Country** has the structure

Country (Name, Capital, Population, Area)

Table 7.2.9 shows some data for table **Country**.

The result set returned when the following SQL query

```
SELECT Name, Capital, Population
  FROM Country
    WHERE (Population < 7000000);
```

is applied to this **Country** table with attributes *Name*, *Capital*, *Population*, *Area* is shown below

El Salvador	San Salvador	5300000
Guyana	Georgetown	800000

Name	Capital	Population	Area
Argentina	Buenos Aires	32 300 003	2777815
Bolivia	La Paz	7 300 000	1098575
Brazil	Brasilia	150 400 000	8511196
Canada	Ottawa	26 500 000	9976147
Chile	Santiago	13 200 000	756943
Colombia	Bagota	33 000 000	1138907
Cuba	Havana	10 600 000	114524
Ecuador	Quito	10 600 000	455502
El Salvador	San Salvador	5 300 000	20865
Guyana	Georgetown	800 000	214969

Table 7.2.9 Table Country showing some values

Questions

4 Write the SQL query that returns the patient surnames from *Table 7.2.3*, for which the patient identifier is less than or equal to 3. Order the result set in descending order of patient identifier (PatientId is the patient identifier).

5 What result set is returned when this SQL query is applied to the data in *Table 7.2.9*?

```
SELECT Capital, Population, Area
  FROM Country
    WHERE (Population > 32000000);
```

Deleting data in a single table

The **DELETE** statement is used to delete rows of a table.

```
DELETE FROM table_name
  WHERE some_column = some_value;
```

The WHERE clause specifies which row or rows should be deleted. If the WHERE clause is omitted, all rows will be deleted!

For example referencing *Table 7.2.9*,

```
DELETE FROM Country
  WHERE Capital = 'Brasilia';
```

deletes the row Brazil, Brasilia, 150400000, 8511196.

Questions

6 Write the SQL statement to delete the row with BorrowerId 3 in the Borrower table shown in *Table 7.2.10*.

7 Write the SQL statement to delete the row(s) with Population > 15000000 in the Country table shown in *Table 7.2.9*.

BorrowerId	Surname	Initial
1	Smith	K
2	Barnes	W
3	Minns	M

Table 7.2.10 Table showing some values for the table Borrower

Inserting data in a single table

The **INSERT INTO** statement inserts a new row into a table. It is possible to write this statement in two forms.

The first form does not specify the column names where the data will be inserted, only their values:

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

The second form specifies both the column names and the values to be inserted:

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

In the first form, a value of the correct data type must be supplied for every attribute of the table and the order of the supplied values must be the same as the corresponding columns in the table.

In the second form, a value for every specified column must be supplied and each value must match in data type the corresponding specified column, i.e. value1 corresponds to column1, value2 to column2, etc. The value Null will be inserted for any columns not referenced.

For example, for table **Ward**, *Table 7.2.2*, reproduced here

WardName	NurseInCharge	NoOfBeds
Victoria	Sister Bunn	30
Aylesbury	Sister Moon	40

Table 7.2.2 Table Ward

First form:

```
INSERT INTO Ward VALUES ('Gresham', 'Mr Oonga', 20);
```

This first form creates a new row in *Table 7.2.2* with values

'Gresham', 'Mr Oonga', 20

Second form:

```
INSERT INTO Ward (WardName, NurseInCharge) VALUES ('Savernake', 'Sister Teng');
```

This second form creates a new row in *Table 7.2.2* with values 'Savernake', 'Sister Teng', Null

Questions

8 Write the SQL statement to add a new row to the Ward table (*Table 7.2.2*) for ward 'Amersham', containing 25 beds. The nurse in charge is 'Sister Brody'.

9 Write the SQL statement to add a new row to the Country table (*Table 7.2.9*) for 'UK', 'London'.

Updating data in a single table

The UPDATE statement is used to update an existing row of a table.

```
UPDATE table_name
  SET column1 = value1, column2 = value2, ...
    WHERE some_column = some_value;
```

For example,

```
UPDATE Ward
  SET NurseInCharge = 'Mr Ali', NoOfBeds = 25
    WHERE WardName = 'Victoria';
```


Questions

10 Write the SQL statement to update the row of the Country table (*Table 7.2.9*) for 'UK' to add population 64100000, area 243610. Assume that an insert statement has inserted 'UK', 'London' already as in Q9.

SQL Tutorials

SQL tutorials are available at https://www.w3schools.com/sql/default.asp.

It is also possible to explore SQL locally by first installing a database engine and then a tool which supports the execution of SQL against a database accessed through the database engine.

SQLite is a self-contained, server-less, zero configuration, transactional SQL database engine. The code for SQLite is public domain and is thus free for use for any purpose, commercial or private. It can be obtained from http://www.sqlite.org/.

An easier route to using SQLite is to download DB Browser for SQLite from https://sqlitebrowser.org/. This application takes care of the installation of both the SQLite database engine and an interface for executing SQL - see *Figure 7.2.1*.

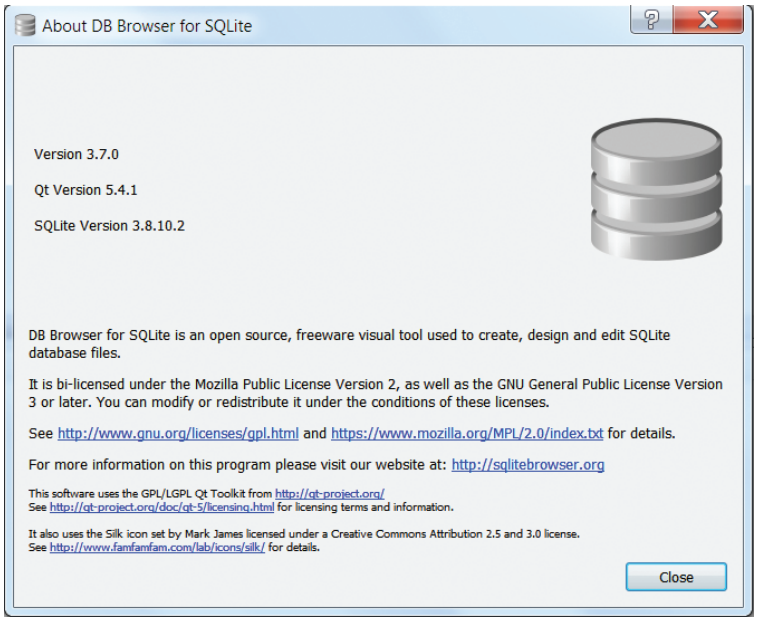

Figure 7.2.1 DB Browser for SQLite

After installing DB Browser for SQLite, launch the application. The user interface for DB Browser for SQLite is shown in *Figure 7.2.2*.

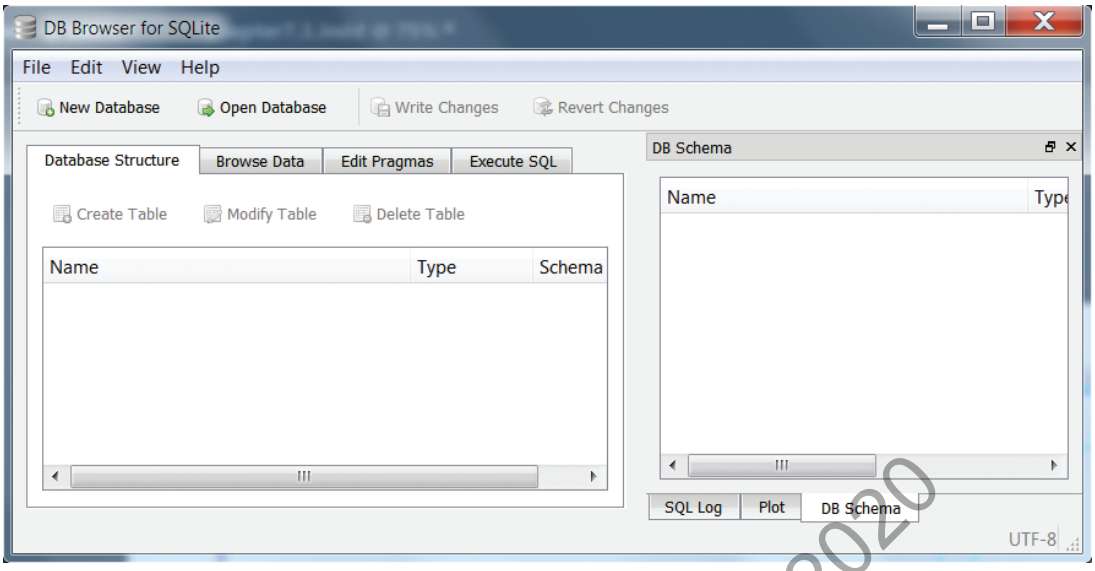

Figure 7.2.2 DB Browser for SQLite user interface

Download the **Hospital.db**, **Country.db**, **Library.db** and **School.db** databases from

www.educational-computing.co.uk/aqacs/gcse8525.html.

Open **Hospital.db** database using the **Open Database** button. *Figure 7.2.3* shows that the opened database consists of two tables **Patient** and **Ward**.

The data stored in the **Ward** table is revealed by executing the SQL query

```
SELECT * FROM Ward;
```

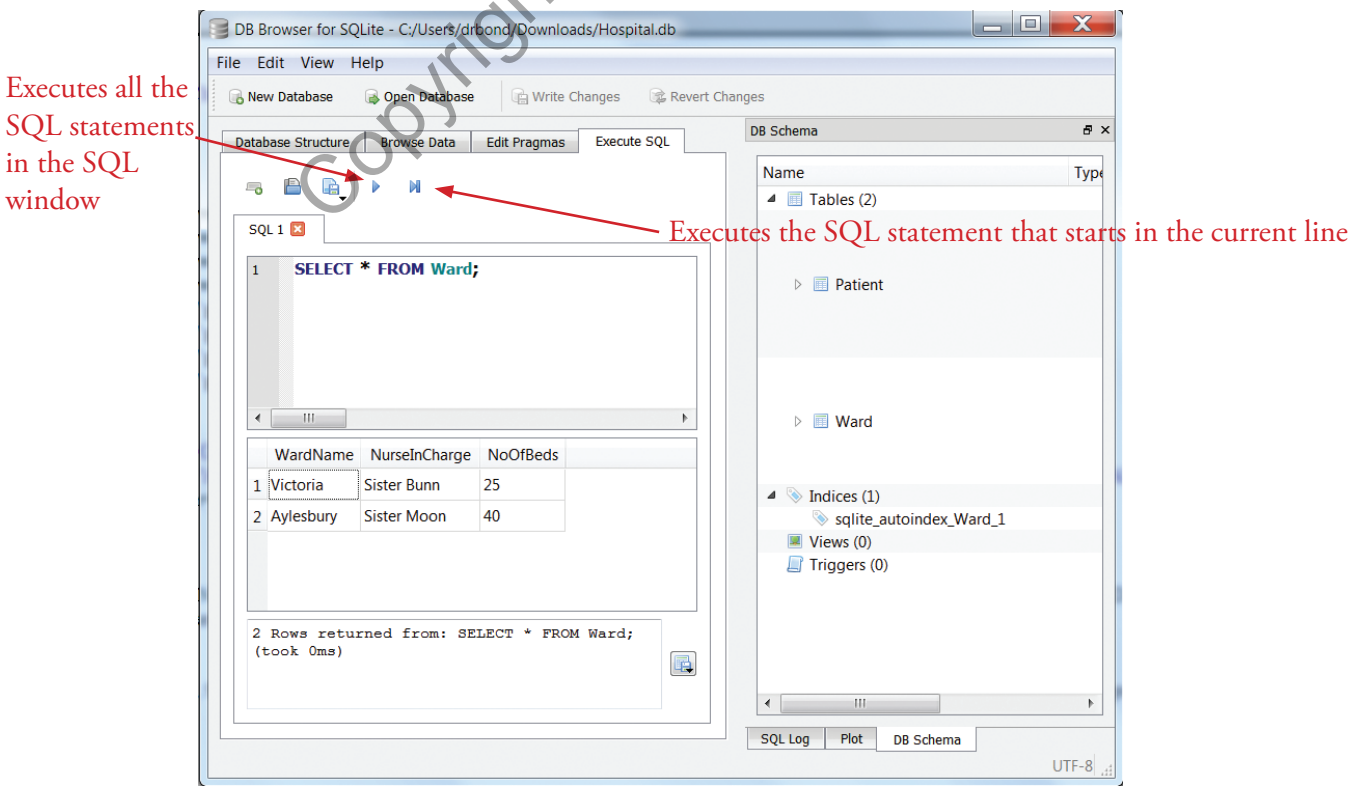

Figure 7.2.3 Execute SQL tab

Figure 7.2.4 shows the result of executing the SQL query

```
SELECT Ward.WardName, NurseInCharge, PatientId
  FROM Ward, Patient
    WHERE Ward.WardName = Patient.WardName
      ORDER BY Ward.WardName ASC;
```

Tasks

1. Try all the SQL examples in this chapter in DB Browser for SQLite.

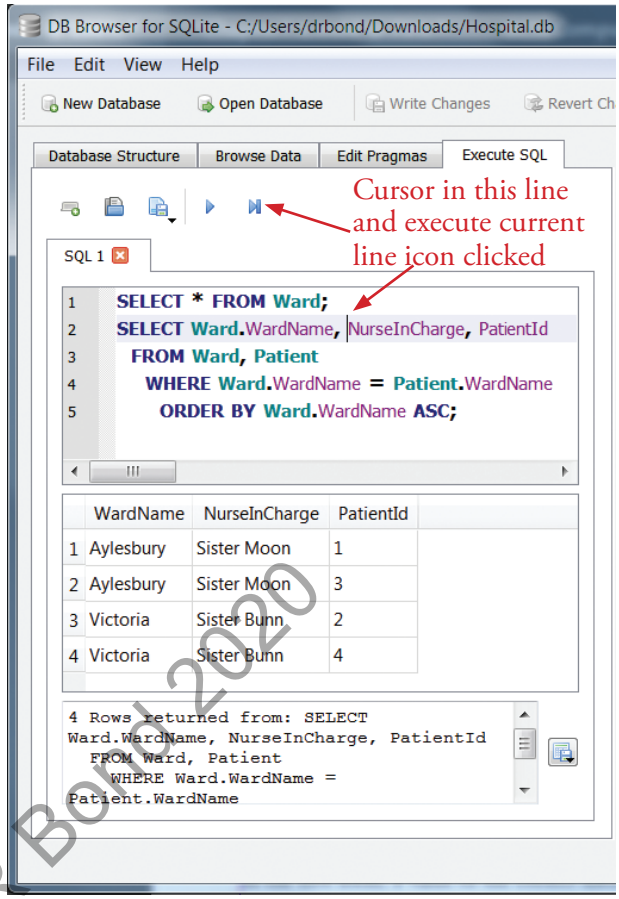

Figure 7.2.4 Querying Ward and Patient tables

In this chapter you have covered:

- How to use SQL to retrieve data from a relational database, using the commands
  - SELECT
  - FROM
  - WHERE
  - ORDER BY...ASC | DESC
- Using SQL to insert data into a relational database by using the command

  INSERT INTO table_name
  (column1, column2, ...)
  VALUES (value1, value2, ...)

- Using SQL to edit and delete data in a relational database by using the commands

  UPDATE table_name
  SET column1 = value1,
  column2 = value2, ...
  WHERE condition

  DELETE FROM table_name
  WHERE condition