Chapter 20 Structured Query Language (SQL)

Contents

Syllabus Learning Outcomes

3.3 Databases and Data Management

Understand, create and use SQL and NoSQL databases, as well as understand techniques to protect the privacy and integrity of data.

3.3.1 Determine the attributes of a database: table, record and field.
3.3.2 Explain the purpose of and use primary, secondary, composite and foreign keys in tables.
3.3.3 Explain with examples, the concept of data redundancy and data dependency.
3.3.4 Reduce data redundancy to third normal form (3NF).
3.3.5 Draw entity-relationship (ER) diagrams to show the relationship between tables.
3.3.6* Understand how NoSQL database management system addresses the shortcomings of relational database management system (SQL).
3.3.7* Explain the applications of SQL and NoSQL.
3.3.8* Use a programming language to work with both SQL and NoSQL databases.

*Note: NoSQL will be addressed in later chapter

1 Structured Query Language

Structured Query Language (SQL) is a standard computer language for the operation and management of relational databases. It is a language used to communicate with a database. SQL can be used to query, insert, update and modify data.

SQL

There are many types of SQL database engines, e.g. MySQL, Microsoft SQL, SQLite, PostgreSQL etc. A database engine is the software that a database management system (DBMS) uses to create, read, update and delete (CRUD) data from a database. For our syllabus, you will be using SQLite.

2 SQLite

SQLite supports the concept of type affinity on columns/fields. Type affinity refers to the preferred data type stored in a column.

Each column in a SQLite table is assigned one of the following type affinities:

Type affinities	Meanings
INTEGER	The value is a signed integer, stored in 0, 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
TEXT	The value is a text string, stored using the database encoding (UTF-8, UTF 16BE or UTF-16LE).
REAL	The value is a floating point value, stored as an 8-byte IEEE floating point number.
NUMERIC	Values entered are converted to any one of the datatypes which is most appropriate.
BLOB	The value is a blob of data, stored exactly as it was input. Usually used to store large binary data, such as images or multimedia in a database.

SQLite does not have a separate Boolean datatype. Instead, Boolean values are stored as integers 0 (false) and 1 (true).

SQLite recognizes the keywords "TRUE" and "FALSE", as of version 3.23.0 (2018-04-02), but those keywords are just alternative spellings for the integer literals 1 and 0 respectively.

Type affinities are relevant in database migrations. This maximizes the compatibility between SQLite and other database management systems. For example, a column declared as

VARCHAR in one database is assigned the type affinity of TEXT if it was migrated to and stored in a SQLite database instead. This feature helps to ensure portability across databases.

In the current syllabus, only INTEGER, TEXT and REAL will be used.

3 Database Operations

In industry-based database applications, SQL commands are grouped into four categories listed below.

- Data Definition Language (**DDL**) defines database schemas.
- Data Manipulation Language (**DML**) is used to retrieve and modify data.
- Data Control Language (**DCL**) is used to control access to a database.
- Transaction Control Language (**TCL**) is used to manage changes to a database, usually at transactional level.

SQL Commands

DDL	DML	DCL	TCL
CREATE	SELECT	GRANT	COMMIT
ALTER	INSERT	REVOKE	SAVEPOINT
DROP	UPDATE		ROLLBACK
RENAME	DELETE		
TRUNCATE	MERGE		
COMMENT	CALL		
	EXPLAIN		
	PLAN LOCK		

	TABLE		

Some of the more advanced commands under DCL and TCL are more relevant to industry specific roles, such as database administrators. For the purposes of our learning, you will need to understand and apply these basic CRUD database operations:

Operation	SQL Command
CREATE	**CREATE, INSERT**
READ (RETRIEVE)	**SELECT**
UPDATE (MODIFY)	**UPDATE**
DELETE (DESTROY)	**DELETE, DROP**

4 DB Browser for SQLite

DB Browser for SQLite is a simple and easy to use Graphical User Interface (GUI) - based software for the creation and editing of database files compatible with SQLite. It abstracts and hides the details of complex SQL commands while providing an easy to use interface for performing the same database operations.

The downloadable version of it is available at http://sqlitebrowser.org/.

An online version of it is known as SQLite Online is found at

https://sqliteonline.com/ # 5 CREATE

In this section, we will explore how to create a library database with four tables (Borrower, Publisher, Book and Loan) using SQL commands.

The library contains books that can be on loan to borrowers.

- A borrower can take one or more loans.
- Each loan record belongs to only one borrower.
- A book can be loaned many times.
- A publisher publishes one or more books.
- A book can be published by zero or one publisher.
  For example, school lecture notes are not published by an official publishing house.

5.1 Create tables

When we create tables, we can set **constraints**. Constraints are the rules enforced on data columns in a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

The following constraints are commonly used in SQL:

● **NOT NULL** – ensures that a field cannot be empty or has a NULL value ● **UNIQUE** - ensures that all values in a column are different
● **PRIMARY KEY** – A combination of a NOT NULL and UNIQUE. Uniquely identifies each row/record in a table.
● **AUTOINCREMENT** – value increases automatically with each new record inserted. ● **FOREIGN KEY** – references to the **PRIMARY KEY** in another table which is used to enforce "exists" relationships between tables. Prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the referenced table

The constraints for **Borrower** table are shown below.

Borrower

Column Name	Type	Constraints
ID	INTEGER	PRIMARY KEY, AUTOINCREMENT
FirstName	TEXT	NOT NULL
Surname	TEXT	NOT NULL
Contact	TEXT	NOT NULL

General SQL statement:

```
CREATE TABLE table_name (
     column1_name COLUMN1_TYPE COLUMN1_CONSTRAINTS,
     column2_name COLUMN2_TYPE COLUMN2_CONSTRAINTS,
     …
     PRIMARY KEY (column1_name, column2_name,…)
     FOREIGN KEY (column1_name) REFERENCES
table_name(column_name)
);
```

SQL statement to create Borrower table:

```
CREATE TABLE Borrower (
     ID INTEGER PRIMARY KEY AUTOINCREMENT,
     FirstName TEXT NOT NULL,
     Surname TEXT NOT NULL,
     Contact TEXT NOT NULL
)
```

OR

```
CREATE TABLE Borrower (
      ID INTEGER AUTOINCREMENT,
      FirstName TEXT NOT NULL,
      Surname TEXT NOT NULL,
      Contact TEXT NOT NULL,
      PRIMARY KEY (ID)
)
```

Let's look at the SQL statement carefully.

INTEGER and TEXT are data types, ID, FirstName, Surname and Contact are field names. The ID field is the primary key of the table Borrower, while Autoincrement means the ID value is automatically given by the database. NOT NULL means that the fields do not accept null values. By default, a PRIMARY KEY field will not accept null values and must be unique.

If a table already exist in the database, creating the same table again will introduce an error saying that the table already exists. Hence, to avoid having such errors, "IF NOT EXISTS" can be included in the CREATE TABLE command.

```
CREATE TABLE IF NOT EXISTS Borrower (
      …
      …
);
```

The statement above will create the `Borrower` table only if `Borrower` table does not already exist.

5.2 Insert data

The INSERT INTO command is used to insert a new record into a table.

General SQL statement:

```
INSERT INTO table_name (column1_name, column2_name,
…)  VALUES (column1_value, column2_value, …);
```

SQL statement to insert a new borrower:

```
INSERT INTO Borrower (FirstName, Surname, Contact)
VALUES ("Peter", "Tan", 99299345)
```

This will insert a new borrower with values for the respective fields into the **Borrower** table.

Note that the **ID** field can be excluded in the **INSERT** command as it is an auto-increment field and will be generated automatically when a new record is inserted into the table.

6 UPDATE

The UPDATE command allows the editing of data values in a database. One or more records may be updated at the same time.

General SQL statement:

```
UPDATE table_name SET
     column1_name = column1_expression,
     column2_name = column2_expression,
     …
WHERE condition;
```

The **WHERE** clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

Given the **Book** table has the following data:

Book table

ID	Title	PublisherID	Damaged
1	The Lone Gatsby	5	0
2	A Winter's Slumber	4	1
3	Life of Pie	4	0
4	A Brief History Of Primates	3	0
5	To Praise a Mocking Bird	2	0
6	The Catcher in the Eye	1	1
123	H2 Computing Ten Year Series	NULL	0

SQL statement to update a record in Book table:

```
UPDATE Book Set Title = 'Book: ' || Title
```

This will update the values of Title in the Book table such that each book title now starts with 'Book: '. Note the use of `||` for string concatenation (adding of two strings).

The updated **Book** table using the **UPDATE** statement above will look like this:

Book table

ID	Title	PublisherID	Damaged
1	Book: The Lone Gatsby	5	0
2	Book: A Winter's Slumber	4	1
3	Book: Life of Pie	4	0
4	Book: A Brief History Of Primates	3	0
5	Book: To Praise a Mocking Bird	2	0
6	Book: The Catcher in the Eye	1	1
123	Book: H2 Computing Ten Year Series	NULL	0

Consider another example using **WHERE** clause.

```
UPDATE Book Set PublisherID = 2 WHERE ID = 3
```

The updated **Book** table will look like this:

Book table

ID	Title	PublisherID	Damaged
1	Book: The Lone Gatsby	5	0
2	Book: A Winter's Slumber	4	1
3	Book: Life of Pie	2	0
4	Book: A Brief History Of Primates	3	0
5	Book: To Praise a Mocking Bird	2	0
6	Book: The Catcher in the Eye	1	1
123	Book: H2 Computing Ten Year Series	NULL	0

7 DELETE

The DELETE command is used to delete existing records in a table.

General SQL statement:

```
DELETE FROM table_name
WHERE condition;
```

SQL statement to delete a record in Book table:

```
DELETE FROM Book WHERE Title = 'The Catcher in the
Eye'
```

This will delete the book ‘The Catcher in the Eye’ from the Books table.

Note: text is case-sensitive, which means that the following 2 SQL statements are different.

```
DELETE FROM Book WHERE Title = 'The Catcher in the Eye'
DELETE FROM Book WHERE Title = 'the catcher in the
eye'.
```

Be careful when using the DELETE statement. If WHERE conditions are accidently left out, the DELETE statement will delete all entries from table.

```
DELETE FROM Book
```

This will delete all entries from the Book table!

That said, DELETE statements only remove records from tables, they do not remove empty tables.
To remove tables, either empty or with contents, DROP TABLE command is used.

General SQL statement:

DROP TABLE table_name

SQL statement to delete Book table:

DROP TABLE Book

This will remove the entire Book table from the database, together with any records in it.

8 READ

8.1 SELECT

The SELECT statement allows the user to retrieve data from the

database. **General SQL statement:**

```
SELECT column1_name, column2_name, …
FROM table_name
WHERE condition
```

To select all fields, use `SELECT *`.

SQL statement to display records in Book table:

`SELECT * FROM Book`
This statement will display all fields of all the books in library.

`SELECT ID, Title FROM Book`
This statement will display the ID and Title fields of all the books in library.

`SELECT * from Book WHERE Damaged = 1`
This statement will display all fields of all the damaged books.

`SELECT * from Book WHERE PublisherID IS NULL`
This statement will display all fields of books with no publishers (no value on the PublisherID field). The IS operator is used to check for NULL values.

More than one conditions can be inserted using AND and OR binary operators.

`SELECT * FROM Book WHERE Title = 'Life of Pie' AND Damaged = 0` This statement returns the books with Title ‘Life of Pie’ **and** are not damaged.

8.2 ORDER BY

The sequence of display of records can also be in either ascending order (default) or descending order of choice.

General SQL statement:

Ascending order

```
SELECT column1_name, column2_name, …
FROM table_name
WHERE condition
ORDER BY order_expression ASC;
```

Note: data retrieved are displayed in ascending order by default, hence, it is optional to include `ASC` in the SQL statement.

Descending order

```
SELECT column1_name, column2_name, …
FROM table_name
WHERE condition
ORDER BY order_expression DESC;
```

SQL statement with ORDER BY:

```
SELECT ID, Title
FROM Book
WHERE Damaged = 1
ORDER BY Title DESC;
```

This would retrieve the ID and Title columns from the Book table, where the book is damaged,

and then order the result set based on the Title column in descending order.

ID	Title
6	The Catcher in the Eye
2	A Winter's Slumber

8.3 GROUP BY

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of books published by each publisher".

The GROUP BY statement is often used with aggregate functions to group the result-set by one or more columns.

Aggregate functions help to count / calculate results from the database.

Aggregate function	Description
COUNT	Number of values
MAX	Maximum value
MIN	Minimum value
SUM	Sum of all values
AVG	Average of values

General SQL statement:

```
SELECT aggregate_func(column_name)
FROM table_name
WHERE condition
GROUP BY column_name
```

SQL statement with GROUP BY:

```
SELECT Damaged, COUNT(ID)
FROM Book
WHERE PublisherID IS NOT NULL
GROUP BY Damaged
```

This query will count the number of books with a PublisherID and group the count according to damaged and not damaged.

Damaged	Count(ID)
0	4
1	2

The output shows that there are 4 non-damaged books with PublisherID and 2 damaged books with PublisherID.

8.4 JOIN

A JOIN allows the combination of data from two sets of data (i.e. two tables). There are different types of joins. We look at three types: **cross join**, **inner join** and **left outer join**.

8.4.1 Cross join

Cross join returns the Cartesian product of rows from the tables in the join. It combines each row in the first table with each row in the second table.

Given that the Publisher table contains 2 fields – `ID` and `Name` with a total of 5 records, `SELECT * FROM Book, Publisher` produces the following table.

ID	Title	PublisherID	Damaged	ID	Name
1	The Lone Gatsby	5	0	1	NPH
1	The Lone Gatsby	5	0	2	Unpop
1	The Lone Gatsby	5	0	3	Appleson
1	The Lone Gatsby	5	0	4	Squirrel
1	The Lone Gatsby	5	0	5	Yellow Flame
2	A Winter's Slumber	4	1	1	NPH
2	A Winter's Slumber	4	1	2	Unpop
2	A Winter's Slumber	4	1	3	Appleson
2	A Winter's Slumber	4	1	4	Squirrel
2	A Winter's Slumber	4	1	5	Yellow Flame
3	Life of Pie	4	0	1	NPH
3	Life of Pie	4	0	2	Unpop

3	Life of Pie	4	0	3	Appleson
3	Life of Pie	4	0	4	Squirrel
3	Life of Pie	4	0	5	Yellow Flame
4	A Brief History Of Primates	3	0	1	NPH
4	A Brief History Of Primates	3	0	2	Unpop
4	A Brief History Of Primates	3	0	3	Appleson
4	A Brief History Of Primates	3	0	4	Squirrel
4	A Brief History Of Primates	3	0	5	Yellow Flame
5	To Praise a Mocking Bird	2	0	1	NPH
5	To Praise a Mocking Bird	2	0	2	Unpop
5	To Praise a Mocking Bird	2	0	3	Appleson
5	To Praise a Mocking Bird	2	0	4	Squirrel
5	To Praise a Mocking Bird	2	0	5	Yellow Flame
6	The Catcher in the Eye	1	1	1	NPH
6	The Catcher in the Eye	1	1	2	Unpop
6	The Catcher in the Eye	1	1	3	Appleson
6	The Catcher in the Eye	1	1	4	Squirrel
6	The Catcher in the Eye	1	1	5	Yellow Flame
123	H2 Computing Ten Year Series	NULL	0	1	NPH
123	H2 Computing Ten Year Series	NULL	0	2	Unpop
123	H2 Computing Ten Year Series	NULL	0	3	Appleson
123	H2 Computing Ten Year Series	NULL	0	4	Squirrel
123	H2 Computing Ten Year Series	NULL	0	5	Yellow Flame

Notice that there are two columns of ID. The first ID column is from Book table, while the second ID column is from the Publisher table.

Aliases can be used to give a column in a table a temporary name to remove the confusion. An alias is created with the AS keyword.

For example,

```
SELECT Book.ID AS BID, Title, PublisherID, Damaged, Publisher.ID AS PID, Name
FROM Book, Publisher
```

BID	Title	PublisherID	Damaged	PID	Name

8.4.2 Inner join and Left outer join

The table above is a big table, with many unnecessary rows (7 x 5 = 35 rows). It is more useful to display all records in Book table with the publisher of each book. To do that, we can add a condition.

```
SELECT * FROM Book, Publisher
WHERE Book.PublisherID = Publisher.ID
```

The SQL statement above gives the following results:

ID	Title	PublisherID	Damaged ID	Name
1	The Lone Gatsby	5	0	5 Yellow Flame
2	A Winter's Slumber	4	1	4 Squirrel
3	Life of Pie	4	0	4 Squirrel
4	A Brief History Of Primates	3	0	3 Appleson
5	To Praise a Mocking Bird	2	0	2 Unpop
6	The Catcher in the Eye	1	1	1 NPH

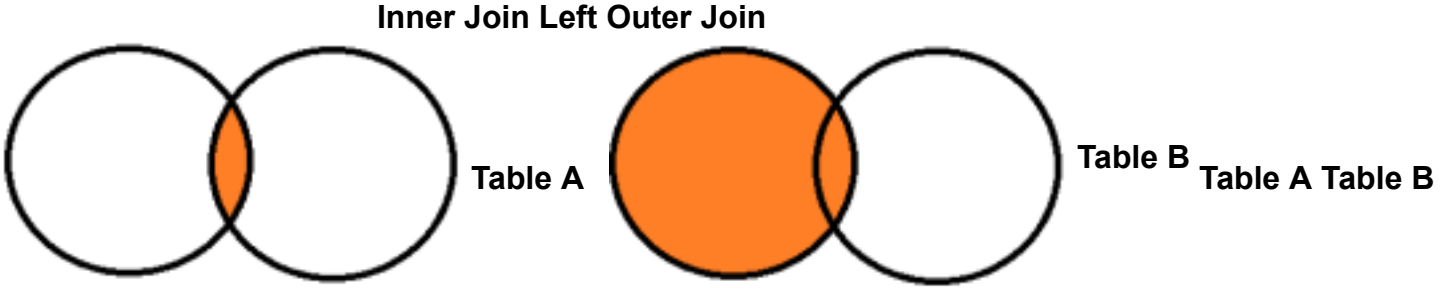
This table is more meaningful as it links the titles to the publishers.

This is an example of inner join. Inner join can also be explicitly defined.

```
SELECT * FROM Book
INNER JOIN Publisher
ON Book.PublisherID = Publisher.ID
```

However, notice that the results only include all books which have publishers. The title ‘H2 Computing Ten Year Series’ is not displayed as it does not have a PublisherID and hence has no publisher.

To display all books with and without a publisher, left outer join can be used.



Inner join selects all records from Table A and Table B which meet the join condition. Left outer join selects all records from Table A, along with records from Table B which meet the join condition.

The results will be different if we change the inner join to left outer join as shown below:

```
SELECT * FROM Book
LEFT OUTER JOIN Publisher
ON Book.PublisherID = Publisher.ID
```

This will display all records in Book table, even books with no publishers, unlike the inner join.

ID	Title	PublisherID	Damaged	ID	Name
1	The Lone Gatsby	5	0	5	Yellow Flame
2	A Winter's Slumber	4	1	4	Squirrel
3	Life of Pie	4	0	4	Squirrel
4	A Brief History Of Primates	3	0	3	Appleson
5	To Praise a Mocking Bird	2	0	2	Unpop
6	The Catcher in the Eye	1	1	1	NPH
123	H2 Computing Ten Year Series	NULL	0	NULL	NULL

References

1. Burdett, A. (2016). *BCS glossary of computing*. BCS Learning and Development Limited. 2. Captain, F. A. (2015). *Six-step relational database design: A step by step approach to relational database design and development*. Place of publication not identified: Publisher not identified. 3. LANGFIELD, S. D. (2015). *Cambridge International AS and A Level Computer Science Coursebook*. Place of publication not identified: Cambridge University Press.
4. Archiveddocs. (n.d.). Lesson 2: Designing a Normalized Database. Retrieved from https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc505842(v=technet.10) 5. Introduction to Database Keys. (n.d.). Retrieved from https://www.studytonight.com/dbms/database-key.php
6. Watt, A. (2014, October 24). Database Design - 2nd Edition. Retrieved from https://opentextbc.ca/dbdesign01/
7. Singh, C., & Prasad. (2018, December 14). Functional dependency in DBMS. Retrieved from https://beginnersbook.com/2015/04/functional-dependency-in-dbms/

8. Chapter 6. Entity-Relationship Modelling. (n.d.). Retrieved from https://www.cs.uct.ac.za/mit_notes/database/htmls/chp06.html#one-to-one-relationships between-two-entities

9. Khan Academy Intro to SQL: Querying and managing data. Retrieved from https://www.khanacademy.org/computing/computer-programming/sql

Annex 1 - SQL Statements

```
CREATE TABLE table_name(
   column1_name COLUMN1_TYPE COLUMN1_CONSTRAINTS,
   column2_name COLUMN2_TYPE COLUMN2_CONSTRAINTS,
   ...
   PRIMARY KEY (column1_name, column2_name, ...),
   FOREIGN KEY (column_name) REFERENCES table_name(column_name)
);
```

```
SELECT column1_name, column2_name, ...
FROM table_name
WHERE where_expression
ORDER BY order_expression ASC;
```

```
SELECT column1_name, column2_name, ...
FROM table_name
WHERE where_expression
ORDER BY order_expression DESC;
```

```
SELECT table1_name.column1_name, table2_name.column2_name, ...
FROM table_name, table2_name
WHERE where_expression;
```

```
SELECT table1_name.column1_name, table2_name.column2_name, ...
FROM table1_name
INNER JOIN table2_name ON join_expression;
```

```
SELECT table1_name.column1_name, table2_name.column2_name, ...
FROM table1_name
LEFT OUTER JOIN table2_name ON join_expression;
```

```
SELECT
   COUNT(*),
   MAX(column1_name),
   MIN(column2_name),
   SUM(column3_name),
   ...
FROM table_name;
```

```
INSERT INTO table_name(column1_name, column2_name, ...)
VALUES(column1_value, column2_value, ...);
```

```
UPDATE table_name SET
   column1_name = column1_expression,
   column2_name = column2_expression,
   ...
WHERE where_expression;
```

```
DELETE FROM table_name
WHERE where_expression;
```

```
DROP TABLE table_name;
```


Annex 2 - Operators

You have seen some operators being used in the examples earlier. These operators are often used in SELECT statements, but can be used in other statements (like the UPDATE statement example shown earlier). The following are comparison operators, logical operators and arithmetic operators that you need to know.

A. Comparison Operators

Comparison Operator	Description
=	Checks if the values of two operands are equal or not, if yes then the condition becomes true.
!=	Checks if the values of two operands are equal or not, if the values are not equal, then the condition becomes true.
<>	Checks if the values of two operands are equal or not, if the values are not equal, then the condition becomes true.
>	Checks if the values of the left operand is greater than the value of the right operand, if yes then the condition becomes true.
<	Checks if the values of the left operand is less than the value of the right operand, if yes then the condition becomes true.
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes then the condition becomes true.
<=	Checks if the value of the left operand is less than or equal to the value of the right operand, if yes then the condition becomes true.

B. Logical Operators

Logical Operator	Description
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS	The value exists. For example, IS NULL looks for NULL values.
IS NOT	The value does not exist.

||	String concatenation

C. Arithmetic Operators

Arithmetic Operator	Name	Description
+	Addition	Adds values on either side of the operator
-	Subtraction	Subtracts the right-hand operand from the left hand operand
*	Multiplication	Multiplies values on either side of the operator
/	Division	Divides the left-hand operand by the right-hand operand
%	Modulus	Divides the left-hand operand by the right-hand operand and returns the remainder

D. Functions

Aggregate functions help to count / calculate results from the database.

Function	Description
MIN	Minimum value
MAX	Maximum value
SUM	Sum of all values
COUNT	Number of values

Annex 3 - Using typeof Function

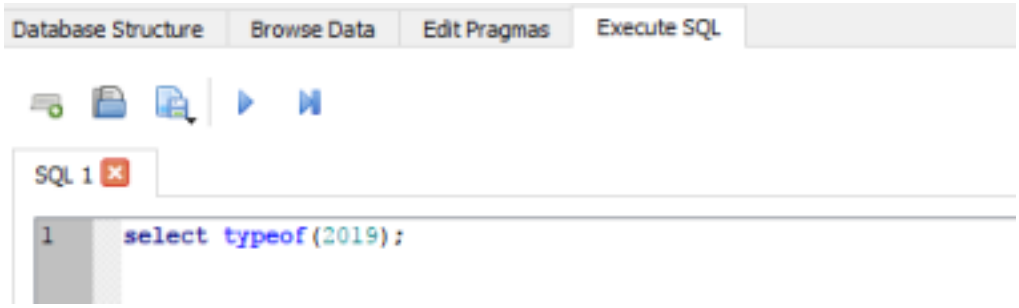
1. Open DB Browser for SQLite.

2. Create a new database.

3. Click on the Execute SQL tab.

4. Enter a simple query using **typeof()** function to find the type for each of the value below.

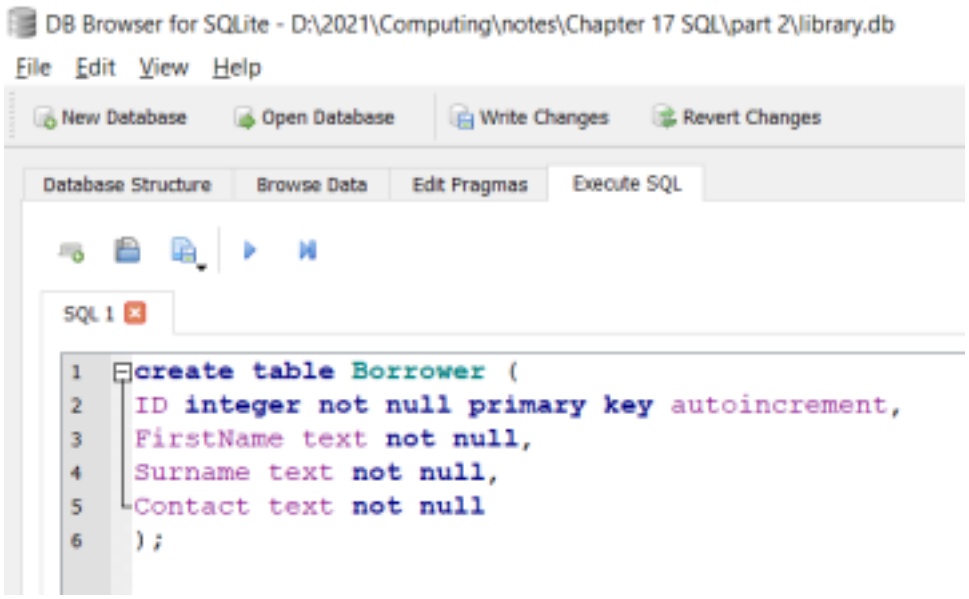
An example of how typeof is used is shown here.



Annex 4 – Using DB Browser to enter SQL statements

1. Open DB Browser for SQLite.
2. Click **File**, then **New Database**.
3. Save your database file as **library**. The default extension is .db.

   Note: other database file extensions are sqlite/sqlite3/db3
4. Click on the Execute SQL tab.
5. Enter the SQL statements to create the Borrower table with the fields and constraints listed. Click ▶ or press F5 to execute the SQL statements.



You can also click on the buttons above the text area.

	Creates a new tab for entering SQL.
	Loads SQL file
	Saves SQL entered as a text file (You may save it as .sql)
	Execute all SQL statements in that tab.

	Execute the current line only.

6. Click **Write Changes** or **CTRL + S** to save changes to the database.

COMMIT in DB Browser for SQLite

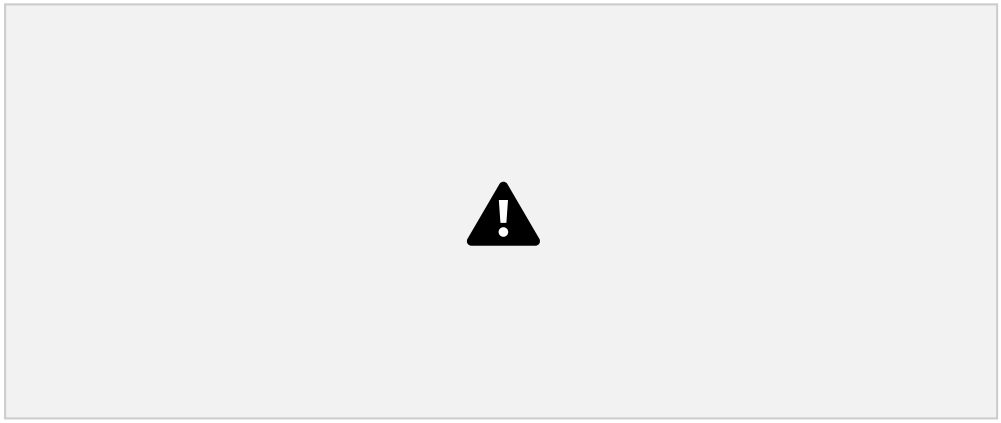
In DB Browser for SQLite, the equivalent of the COMMIT command in SQLite is **Write Changes.** This feature saves changes but does not close the database file.

You can also click on **Revert Changes** to undo changes made to the database.

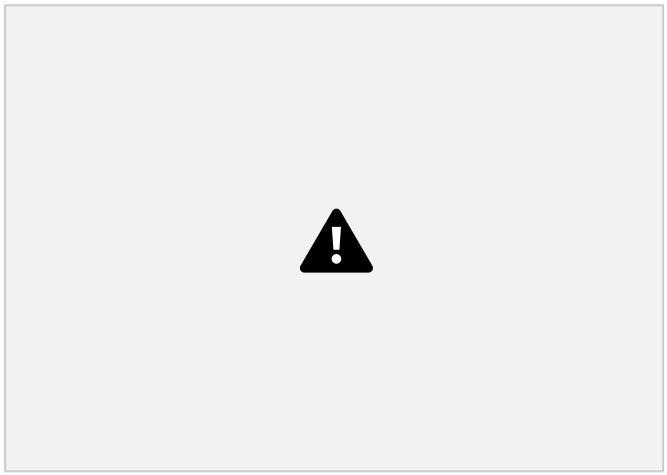
Annex 5 – Using Import table from CSV file

1. Create the **Loan** table using the **Import** feature.

This feature also allows importing of .TXT and .CSV files.

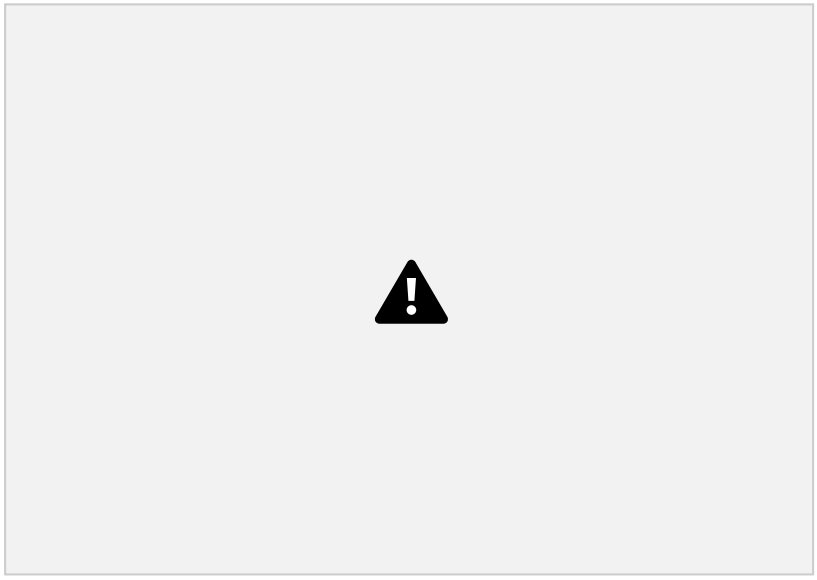


2. Select Loan.txt.



3. Click **Open**.

4. Tick the option **Column names in first line**.



5. Click OK.
6. Click **Modify Table**.
7. Edit the types according to the description above.

Note:
The types for every column in the table are defaulted to TEXT during an import. Hence it is important that you check on the types after an import.

8. Tick the constraints as according to below.

Table Constraints:

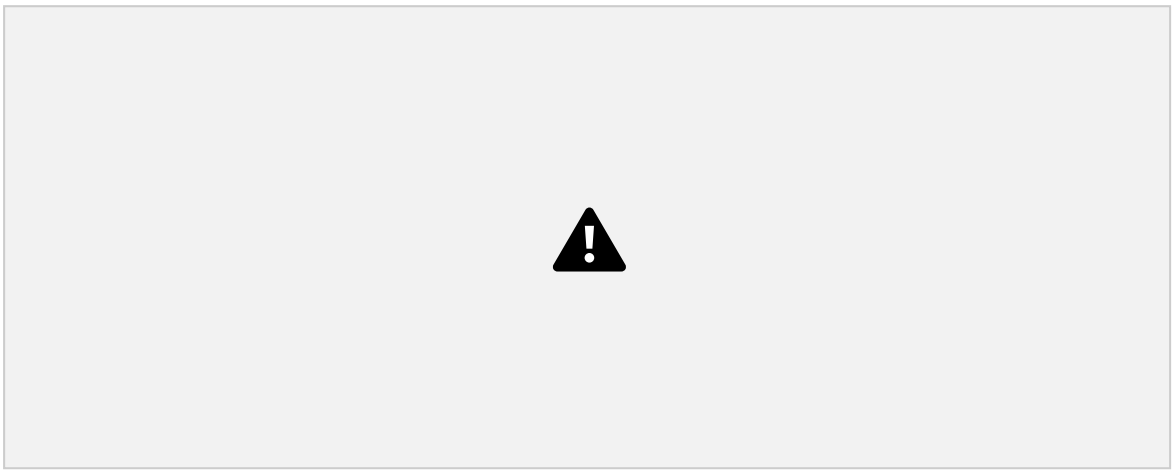
● ID is the **PRIMARY KEY** of the Loan table
● The value of ID should be **AUTOINCREMENT**
● ID, BorrowerID and BookID fields are **NOT NULL**

For BorrowerID and BookID of the Loan table, identify the **FOREIGN KEY** constraints.

● BorrowerID is a **FOREIGN KEY** to ID in the Borrower table ● BookID is a **FOREIGN KEY** to ID in the Book table.

9. To create the foreign key for BorrowerID, highlight the BorrowerID attribute by clicking on the row.
   Choose **Borrower** and **ID** from the drop down list under Foreign Key

   column. This creates a foreign key reference to ID in the Borrower table.



10.Repeat the above step for BookID to create the foreign key reference. 11.View the Loan table from **Browse Data** tab. You should see the following data:

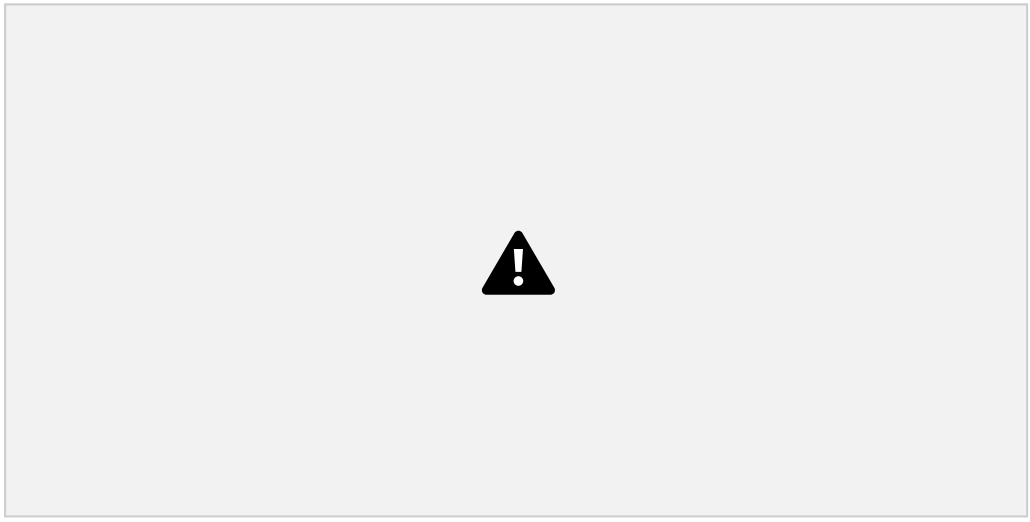
ID	BorrowerID	BookID	DateBorrowed
1	3	2	20180220
2	3	1	20171215
3	2	3	20171231
4	1	5	20180111

12.Write changes to the database.

Annex 6 – Using Import database from SQL file

1. Create the **Loan** table using the **Import Database from SQL file** feature

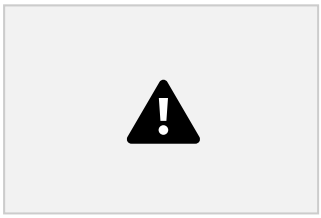
   This feature also allows importing .SQL files.



2. Select “insert load.sql”

3. Select **No** to import data into the current database that is opened (library.db)

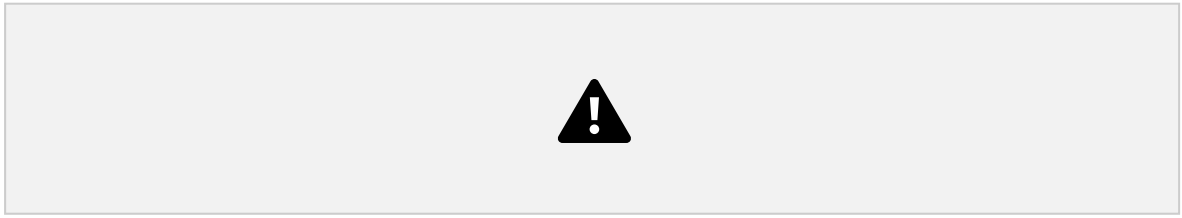
4. If SQL file is imported successfully, you will see the import completed message.



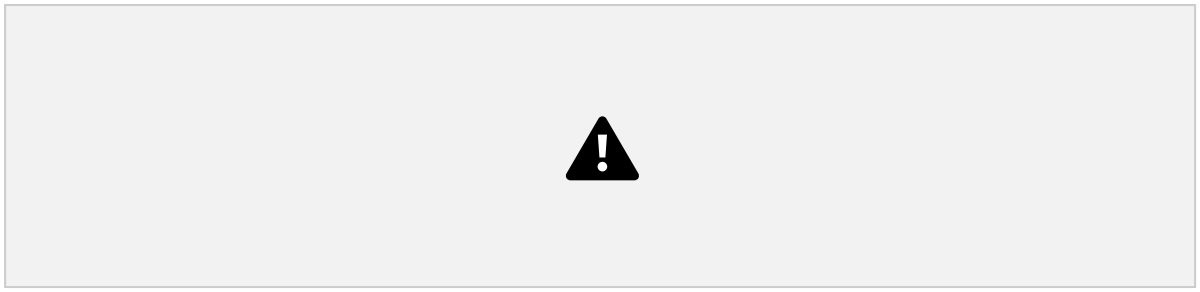
5. Write changes to the database.

Annex 7 - Insert Records using GUI

1. Under the Browse Data tab, click **New Record**.



2. Click on the FirstName cell of the first record.



3. Under Edit Database Cell, type the value for FirstName. Click **Apply**.

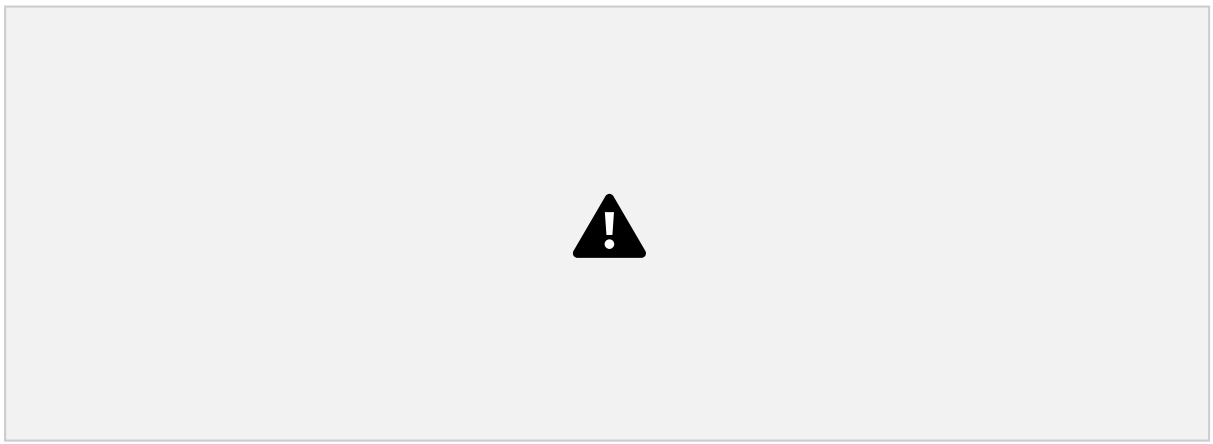
4. Repeat the above two steps for Surname and Contact.

If the record has been entered correctly, you should see the following in the table: 

5. Write changes to the database.

Annex 8 - Update Records using GUI

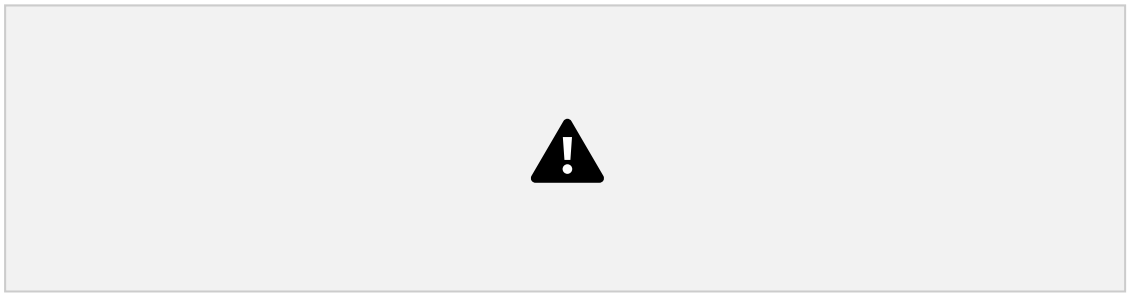
1. Click on the Contact cell of the record you wish to update.



2. Under Edit Database Cell, type in value to be updated.
3. Click **Apply**.

Annex 9 - Delete Records using GUI

1. Select record to be deleted.



2. Click **Delete Record**.
3. Write changes to the database.

Annex 10 - **Revert Changes**.

ROLLBACK in DB Browser for SQLite

In DB Browser for SQLite, the equivalent of the ROLLBACK command in SQLite is **Revert Changes.** This feature is useful for undoing any modifications to the database since the last saved state.

Annex 11 - Summary of DB Browser for SQLite Features

File Features

Name	Description
	Creates a new SQLite database file. File extensions are .db or .db3 or .sqlite or .sqlite3
	Opens a SQLite database file.
	Saves changes to database. Equivalent to COMMIT operation.
	Undo any changes made to the database. Equivalent to ROLLBACK

	operation.
	Imports either a CSV or TXT file as a table into the database or SQLite database file.
	Exports one of the following: ● a table in the database as a new CSV file ● database as a SQLite file ● database as a JSON file

Database Structure

Name	Description
	Creates a table in the database.
	Modifies a table in the database, not limited to changing table name or field name, adding fields or constraints.
	Deletes a table from the database.

Constraints are the rules enforced on data columns or table. These are used to limit the type of data that can go into a column or table. This ensures the accuracy and reliability of the data in the database. Constraints could be at a column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

Constraints

Name	Description
	Ensures that a column cannot have NULL value.
	Sets a PRIMARY KEY constraint such that it uniquely identifies each row/record in a table.

	Automatically increments the value of the attribute for each new record. Works for integer values only.
	Ensures that all values in a column are unique.
	Provides a default value for a column when none is specified.
	Ensures that all values in a column satisfies certain conditions. If the condition evaluates to false, the record violates the constraint and isn't entered into the table.
	Sets a FOREIGN KEY constraint where a column of a table can reference a column from another table or within the same table.

Browse Data

Name	Description
	Refresh data in the selected table.
	Clear all filters.
	Creates a new record in the table.
	Deletes a record in the table.