Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

LEAVING CERTIFICATE
COMPUTER SCIENCE

# Fundamental Skills Development

*Databases*

pdst.ie

# Manual Overview

The purpose of this manual to provide Phase One Leaving Certificate Computer Science (LCCS) teachers with the knowledge, skills and confidence to independently design and develop websites and web applications.

Although the manual will serve as support material for teachers who attend the Web Application Development Workshop component of our two-year CPD programme, it is envisaged that its real value will only become evident in the months after the workshops have been delivered. Beyond these workshops, the manual may be used as a basic reference for web development, but more importantly, as a teaching resource that might be used to facilitate teachers in employing a constructivist pedagogic orientation towards the planning for teaching and learning of web development in the LCCS classroom.

The manual itself is divided into five separate sections and is split into three separate documents – Part A, Part B and Part C – organised as shown below. This is Part C.

**Part A**

**Section 1 – HTML**

**Section 2 – Cascading Style Sheets**

**Section 3 – UX Design**

**Part B**

**Section 4 – JavaScript**

**Part C**

**Section 5 – Databases**

# Section 5

## Databases

## Introduction to Databases

- History and Future of databases
- Data v Information / Big data / Database and DBMS
- Information Systems (Activity: Ireland Hockey team / School MS)
- Single table database – row, column, data types, primary key
- Creating Tables, Queries, Forms, Reports in *OpenOffice Base*
- Relational databases – relationships
- Examples / Exercises: Class Library, Tennis Club

## Introduction to Structured Query Language

- Syntax and Semantics
- Creating database from command line in *MySQL, Sqlite*
- Insert, Select, Update and Delete data
- Examples / Exercises: 'Hodson Bay' / Tennis club / Class Library

## Using Web Technologies with Databases

- Uploading to web server and accessing
- Creating webpage linking to *Base* db
- Using an interactive web front end with html and JavaScript
- Examples / Exercises: Class library / Wedding Presents.

# Databases and Information

It's only a short space of time since a database, possibly just a stand-alone single table or relational database was just to store data and to retrieve, query and manipulate data into useful information. This may have been the norm for a school, a doctor's or dentist's practice or a small busines. With the advance of the web and of storage capabilities, the centrality of the database has diminished, and Information Systems have become more advanced and more integrated. These advances have ensured that raw unstructured data can be used and are being used as the source.

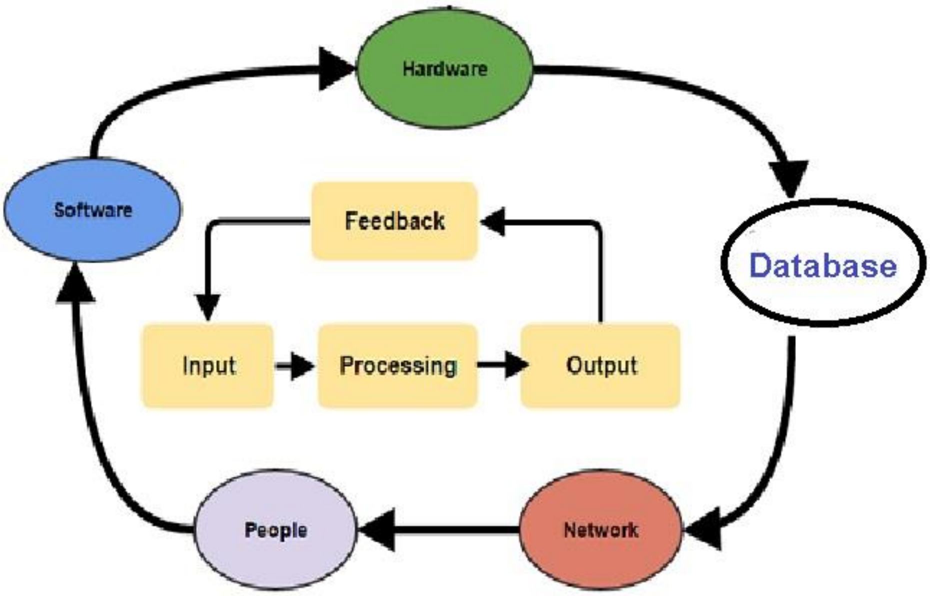
When students are asked what are Information Systems, they often answer with short answers such as "computers", "Excel" or "databases". But Information Systems in some of the contexts above have changed.

An Information System is software that can organise and analyse data. Consider the systems in schools and colleges which have information about:

- Teachers' and Students' Timetables
- Teachers' class groups
- Students' academic record
- Facility for Teachers to enter data on Students' performance and behaviour
- Bookings for labs, libraries, computer rooms

Behind the one front end there are multiple files some of which are databases.

Fig 1:  General Information Systems with a Database:




With the advance of Information Systems, the role of the database has changed.  The advance of **big data** means a huge amount of data can be analysed and mined from 'flat files'.

But what makes something a **database**?  It's something that contains data which has a structure so that the data are organised and can be managed, accessed and updated.

We'll consider several different types of stand-alone databases, and how to perform the actions mentioned above, on them.

Firstly, however, let's take a look at the bigger picture.  Consider an example of a large organisation where data and information flow to and from management through various channels and to and from various entities.  The information then helps management in decision-making, administration, planning, leading and improving.

## Example 1: Irish Women's Hockey Team – Management Information System:

The Ireland team was very successful at last year's World Cup. Consider the Information required and used by the team management. Everything involved with the players, coaches, other staff, other teams, travel and expenses, dealing with the hockey federation etc.

**Fig 2: Diagram showing flow of data and Information to and from Irish Hockey team management:**

## Exercise 1: School Management System

Now try do so something for a management system for your school placing the management team of principal and deputy principal at the centre.

**Diagram:**

It is also useful to define a **database management system** (DBMS). DBMS is the software which creates and manages databases in a systematic way.

**DBMSs** should display *ACID* properties (Atomicity, Consistency, Isolation, Durability) and can be categorised into:

- Relational Database Management Systems (RDBMS) such as
  - Oracle Database
  - MySQL
  - PostgresSQL
  - MS_SQL
  - Sqlite
  - MS Access
  - IBM DB
  - MS SQLServer

- Document Store (aka No SQL) – Non-relational, schemaless databases:
  - MongoDB
  - CouchDB
  - MarkLogic
  - DocumentDB
  - ZODB

- Graph databases: Interconnected records of the same or similar type. For example, answering the question “Who are the friends of my friends?” in Social Networks:
  - Neo4J
  - ArangoDB
  - OrientDB
  - GraphDB

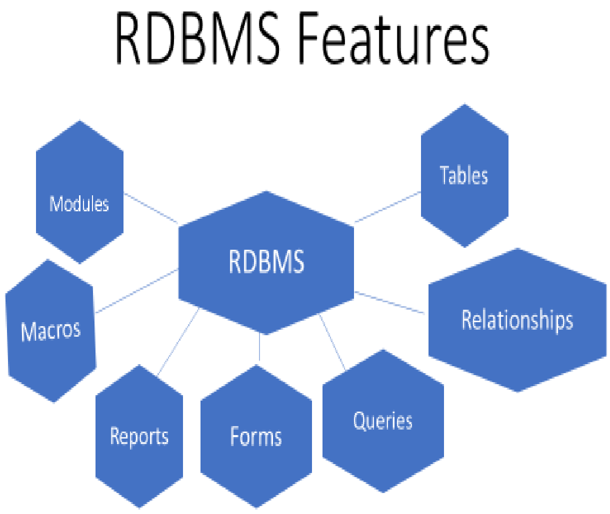
- Column Store – organised by columns rather that rows:
  - Apache HBase
  - PostgresSQL
  - MariaDB column store

- Key Value Store – databases store data indexed by a unique key:
  - Redis
  - BerkleyDB
  - Riak

**Relational Databases** with tables and relationships between them were often the choice of schools, small companies and organisations, as they featured build-in queries, forms, report writing capability and macros and modules for more advanced users. They started with EF Codd of IBM, based on the principles:

- No duplicate data
- Information broken down into categories
- Data broken down into the smallest useable bit: eg ‘Name’ could be broken down into ‘Title’, ‘FirstName’, ‘MiddleName’, ‘Surname’.

**Fig 3: Features of RDBMS:**




We'll now look, through examples, and exercises, at the process of designing and using databases, to include. The first examples / exercises use *OpenOffice's Base* application (downloadable at www.openoffice.org):

- Planning a database
- Creating a new database
    - Creating table(s) using design view / Wizard
    - Adding Data
    - Creating Database form(s)
    - Querying tables
    - Creating Reports
    - Accessing spreadsheets
    - Creating relationships where more than one table is used.

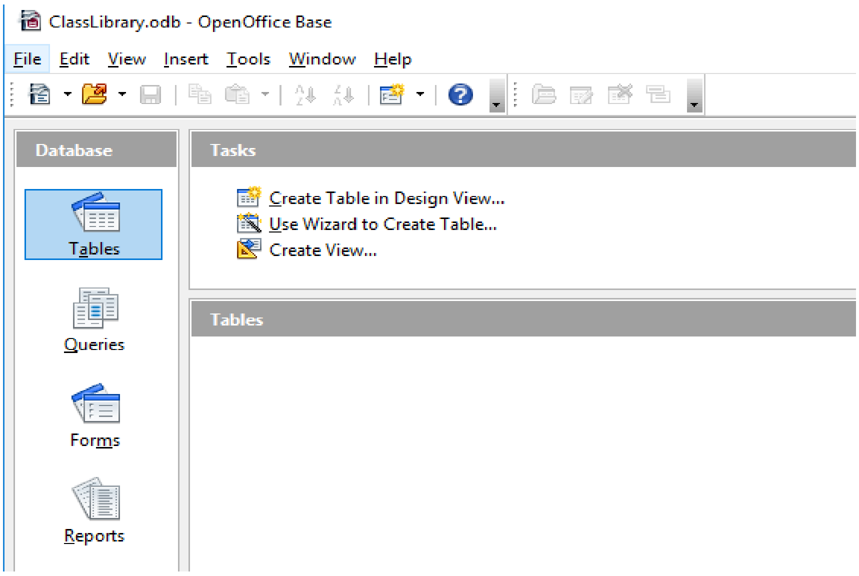
## Example 2: Designing a database system for a Class Library

In this first example, we will design a system with one table for a class library. As it only has one table, the first step is to decide on the fields.

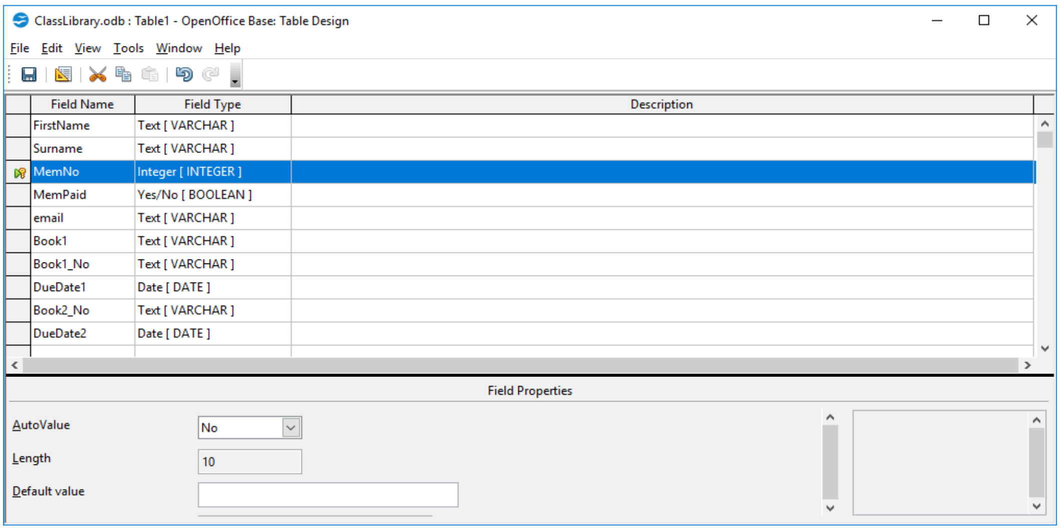
I'm presuming each student will be able to take out 2 books at a time for two weeks, so the name of the books and the book number will be required.

In addition, there will obviously have to be details about the members, such as name, email address, and member number. As mentioned before, name can be broken down into first and surname.

So we'll have (**Fig 4**):



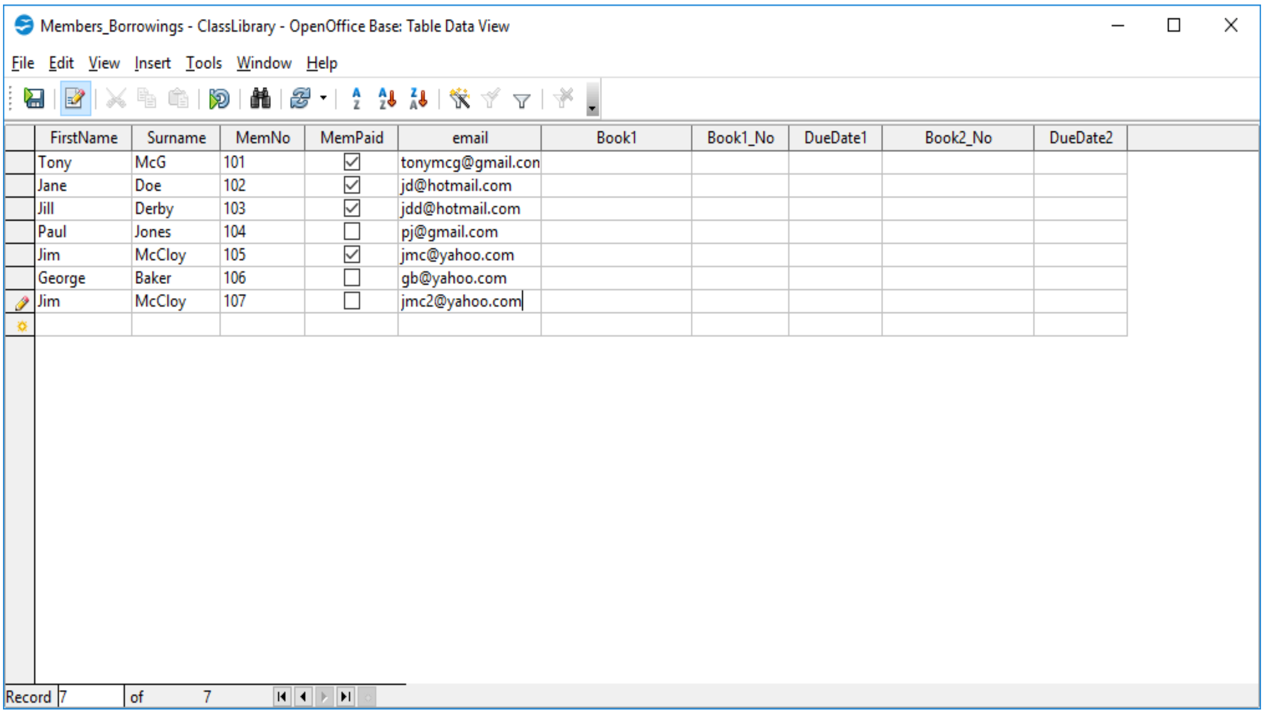

From the three options, "Create Table in Desisgn View" was chosen to give (**Fig 5**):



I decided on the 10 fields, and chose the appropriate field type – eg a text field with variable number of characters for Surname, and a boolean *yes / no* for MemPaid, to record if the students paid an initial small sign-up fee for the service. You'll notice a field 'Book1_No' in addition to 'Book1' to allow for having, say, three copies of 'Treasure Island'.

Also note the highlighted field MemNo, which has been chosen as the Primary Key, and each member will have a unique and distinct membership no. The Primary key is also essential in linking tables.

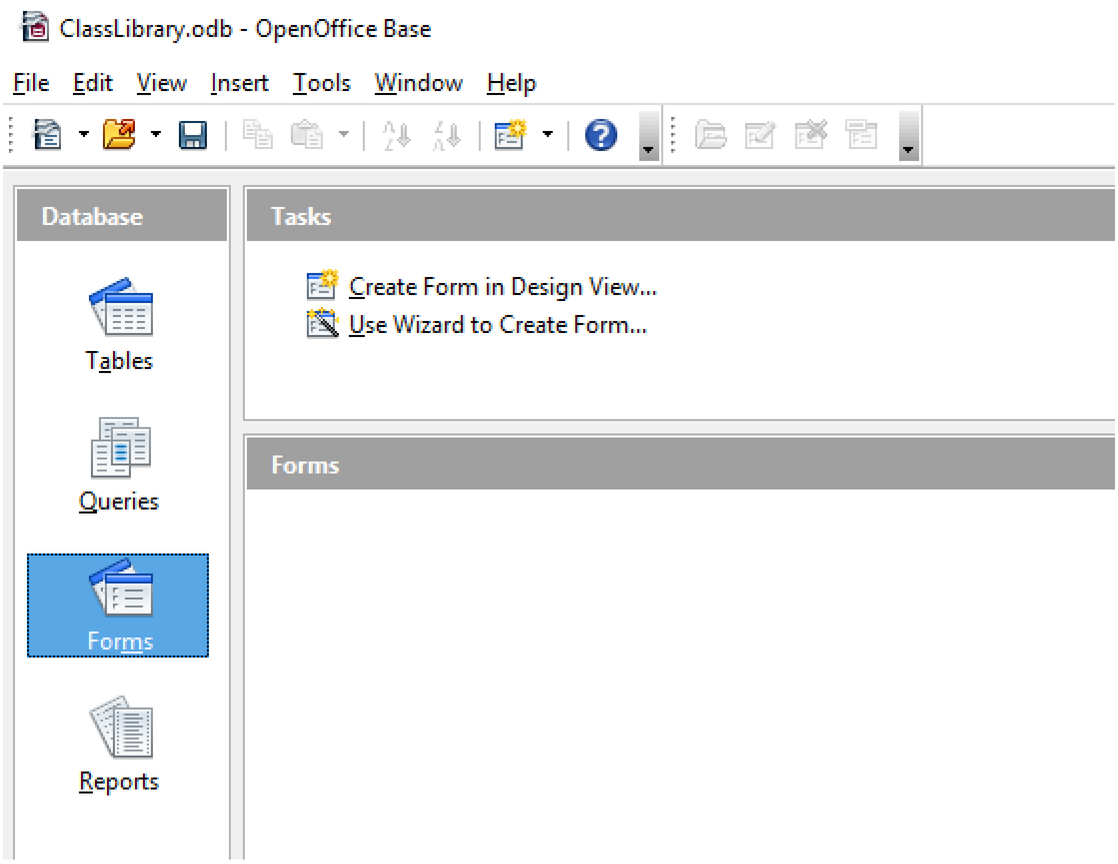
The next step is to enter some data into the table (**Fig 6**):

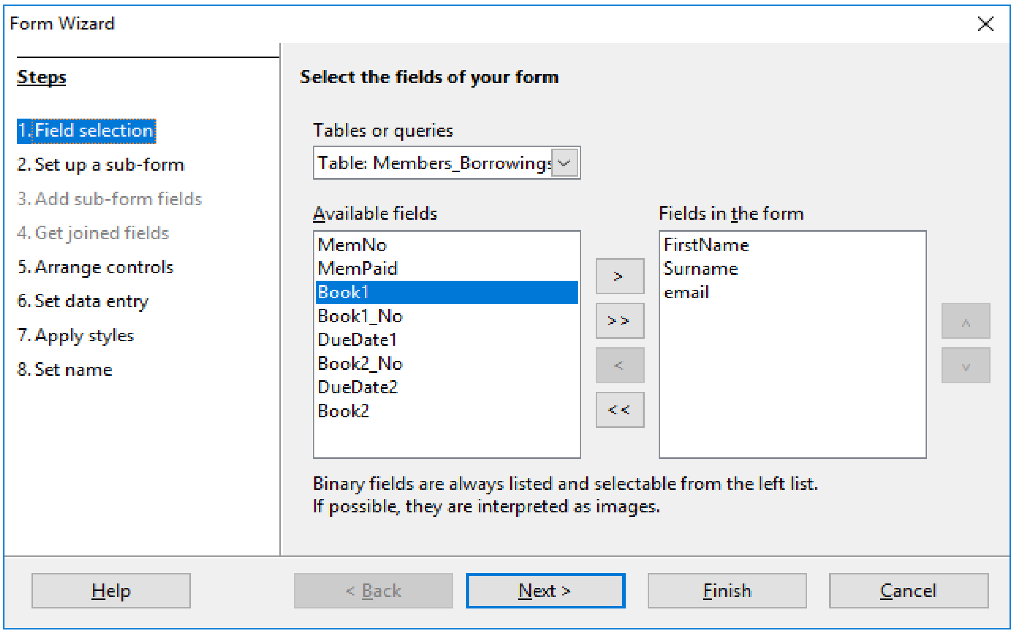



| FirstName | Surname | MemNo | MemPaid | email | Book1 | Book1_No | DueDate1 | Book2_No | DueDate2 |
|---|---|---|---|---|---|---|---|---|---|
| Tony | McG | 101 | ☑ | tonymcg@gmail.con | | | | | |
| Jane | Doe | 102 | ☑ | jd@hotmail.com | | | | | |
| Jill | Derby | 103 | ☑ | jdd@hotmail.com | | | | | |
| Paul | Jones | 104 | ☐ | pj@gmail.com | | | | | |
| Jim | McCloy | 105 | ☑ | jmc@yahoo.com | | | | | |
| George | Baker | 106 | ☐ | gb@yahoo.com | | | | | |
| Jim | McCloy | 107 | ☐ | jmc2@yahoo.com | | | | | |

As you can see here, the Primary key distinguishes Member No.107 from Member No.105, although both are called Jim McCloy.

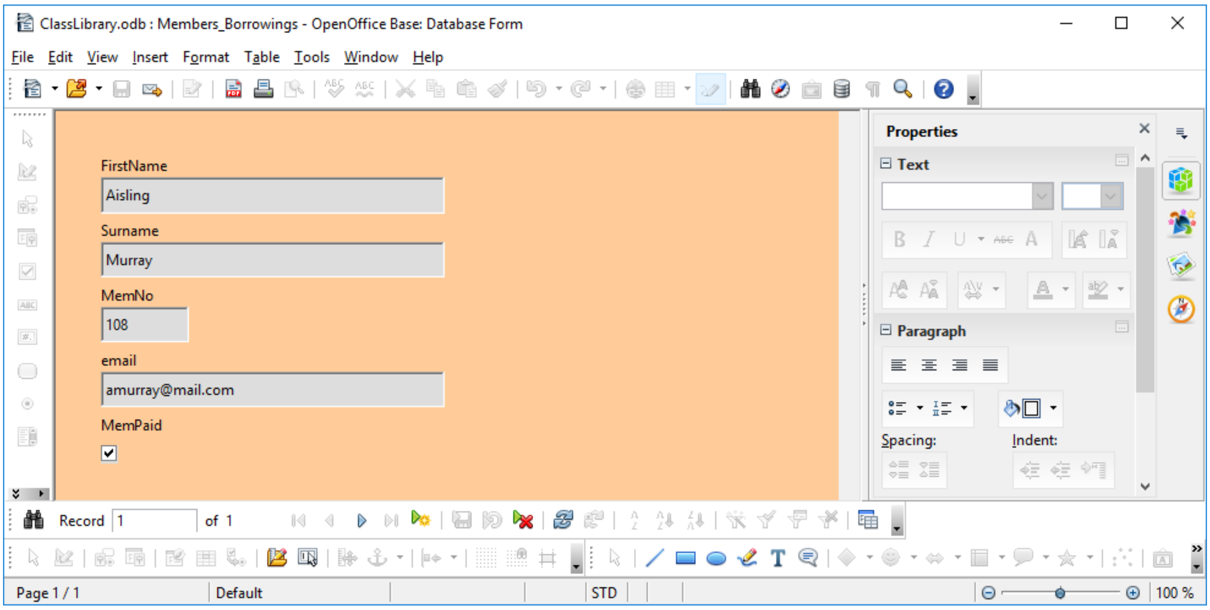
We can now create a form using Form Wizard (**Fig 7**):



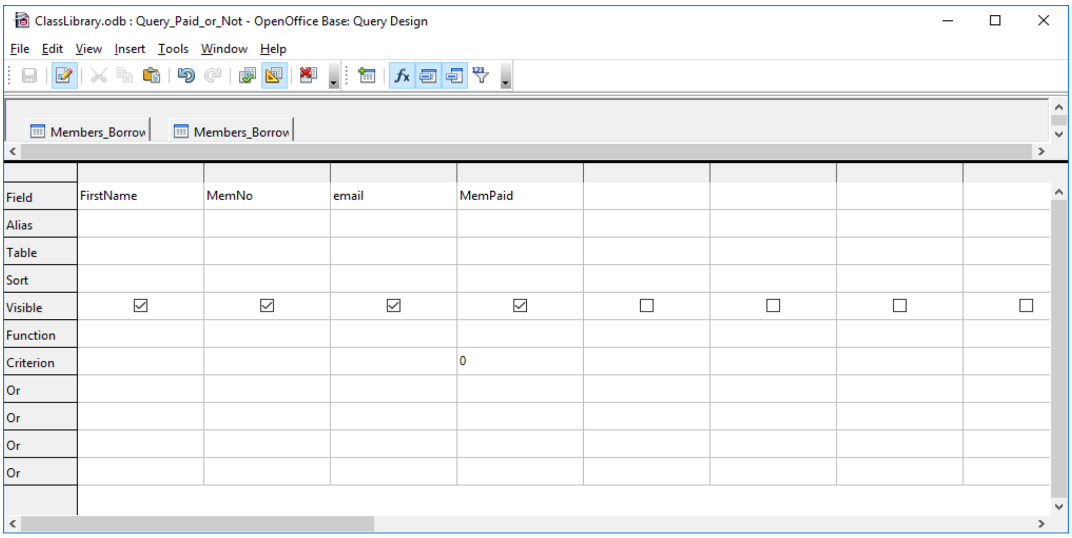

This screen shot shows the fields you can select for your form.

I chose FirstName, Surname, MemNo, Email and MemPaid to produce the final form (**Fig 8**):

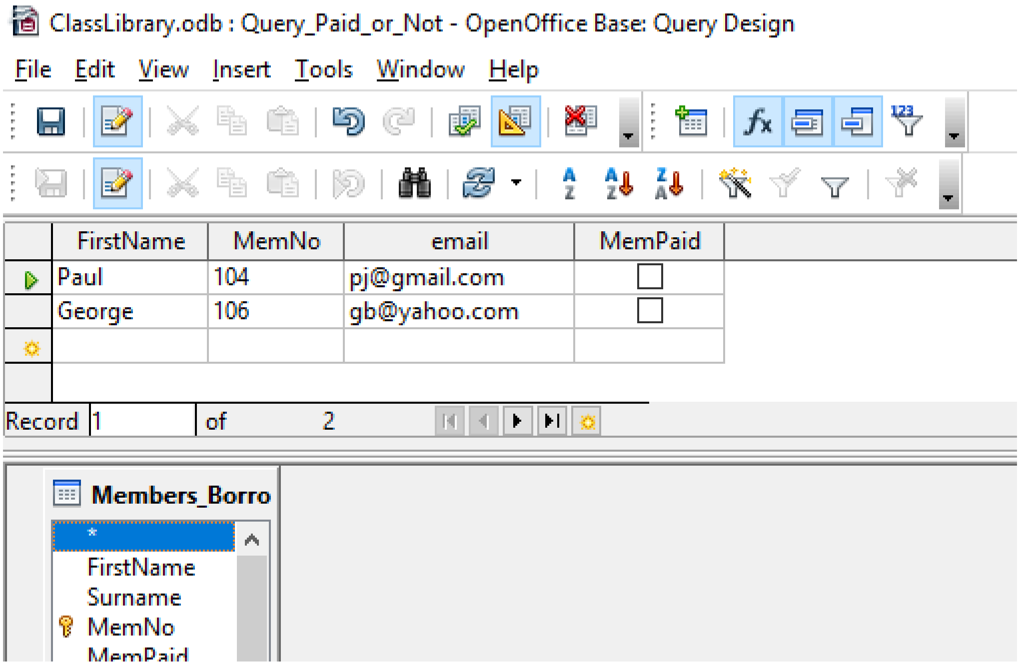



The form is used to add records to out table. Be careful that the fields you choose can be null, or you'll get an error message.

Queries are used to access some of the data from a database and are a very important aspect of database functionality. Suppose we wanted to have a list of those who haven't paid in order that we can send them an email reminder (**Fig 9**):

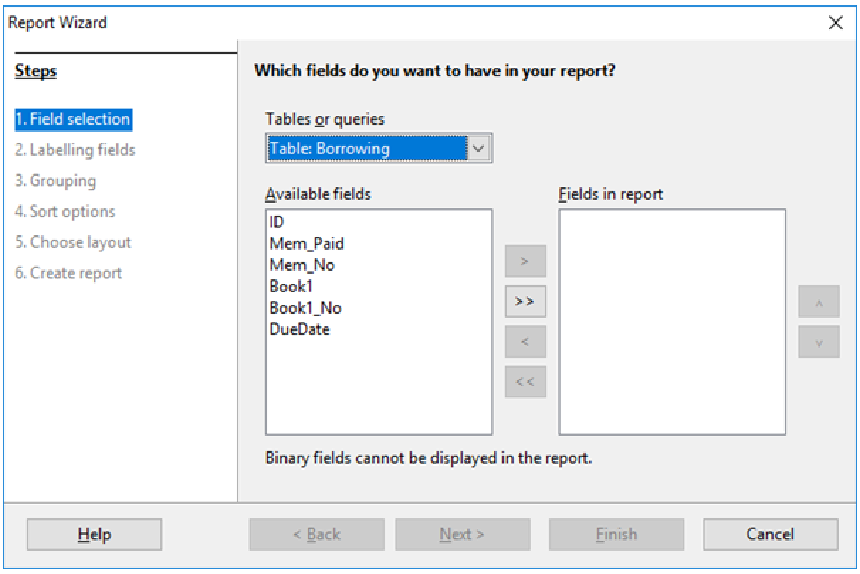


The chosen fields are there and the Criterion for the Mem Paid field is “=0”, to show that the value is this Field is “No”, as in “not paid”.  The results of the query are here (**Fig 10**):




This shows, as expected, that Paul (104) and George (106) have not paid.

We may want to produce a **Report** of some of the data.  Suppose we just wanted the first name, surname and membership number for each member.  This time I used the Wizard to select the required fields (**Fig 11**):



This produced the following report (**Fig 12**):

Members_Borrowings - OpenOffice Base: Report Design

sert Format Table Tools Window Help

Title:

Author: Tony McGennis
Date: 14/05/19

| *FirstName* | *Surname* | *MemNo* |
|---|---|---|
| Tony | McG | 101 |
| Jane | Doe | 102 |
| Jill | Derby | 103 |
| Paul | Jones | 104 |
| Jim | McCloy | 105 |
| George | Baker | 106 |
| Aisling | Murray | 108 |

Now let's add in data for some book borrowings (**Fig 13**):

Members_Borrowings - ClassLibrary - OpenOffice Base: Table Data View

File Edit View Insert Tools Window Help

| FirstName | Surname | MemNo | MemPaid | email | Book1 | Book1_No | DueDate1 | Book2_No | DueDate2 | Book2 |
|---|---|---|---|---|---|---|---|---|---|---|
| Tony | McG | 101 | ☑ | tonymcg@gmail.con | Treasure Island | 201 | 10/05/19 | 230 | 10/05/19 | Hard Times |
| Jane | Doe | 102 | ☑ | jd@hotmail.com | Treasure Island | 202 | 15/05/19 | | | |
| Jill | Derby | 103 | ☑ | jdd@hotmail.com | | | | | | |
| Paul | Jones | 104 | ☐ | pj@gmail.com | Papillon | 288 | 25/05/19 | | | |
| Jim | McCloy | 105 | ☑ | jmc@yahoo.com | | | | | | |
| George | Baker | 106 | ☐ | gb@yahoo.com | | | | | | |
| Aisling | Murray | 108 | ☑ | amurray@mail.com | | | | | | |

Record 2 of 7

And if we want to find out the names of students who had books taken out we could run the following query (**Fig 14**):

| | | | | | | |
|---|---|---|---|---|---|---|
| Field | FirstName | Surname | Book1_No | Book1 | Book2 | Book2_No |
| Alias | | | | | | |
| Table | Members_Borrowings | Members_Borrowings | Members_Borrowings | Members_Borrowings | Members_Borrowings | Members_Borrowings |
| Sort | | | | | | |
| Visible | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ |
| Function | | | | | | |
| Criterion | | | NOT '0' | | | |
| Or | | | | | | |
| Or | | | | | | |

The query would produce these results (**Fig 15**):

| FirstName | Surname | Book1_No | Book1 | Book2 | Book2_No |
|---|---|---|---|---|---|
| Tony | McG | 201 | Treasure Island | Hard Times | 230 |
| Jane | Doe | 202 | Treasure Island | | |
| Paul | Jones | 288 | Papillon | | |

Record 1 of 3

| | | | | | | |
|---|---|---|---|---|---|---|
| Field | FirstName | Surname | Book1_No | Book1 | Book2 | Book2_No |
| Alias | | | | | | |
| Table | Members_Borrowings | Members_Borrowings | Members_Borrowings | Members_Borrowings | Members_Borrowings | Members_Borrowings |
| Sort | | | | | | |
| Visible | ☑ | ☑ | ☑ | ☑ | ☑ | ☑ |
| Function | | | | | | |
| Criterion | | | NOT '0' | | | |

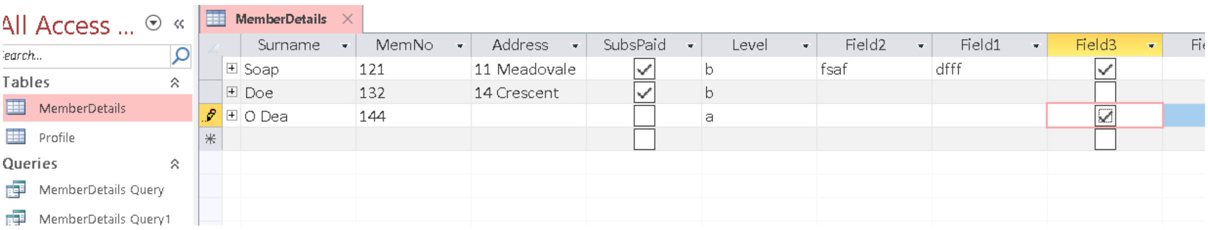
## Exercise 2:  Try some more data entry and queries:

1. Add three more records to the table, using forms.
2. Include some borrowings for these new records.
3. Design a query which gives the first and last name of those who have no books out.
4. Design a query which gives the the first name, surname and the names of the books for those who have taken out two books.

## Example 3: Extending the Class library to include multiple tables wth links between them.

The term ‘Relational databases’, as was mentioned above, implies that there is a relationship between tables.  If we only use one table, we are in danger of having too many fields.  The resulting table can look unwieldy and may be difficult to manipulate:

Fig **16a**:  Database table with 21 fields:




This table has actually has 21 fields, but only 9 are visible on one screen – to view the others we need to scroll across.

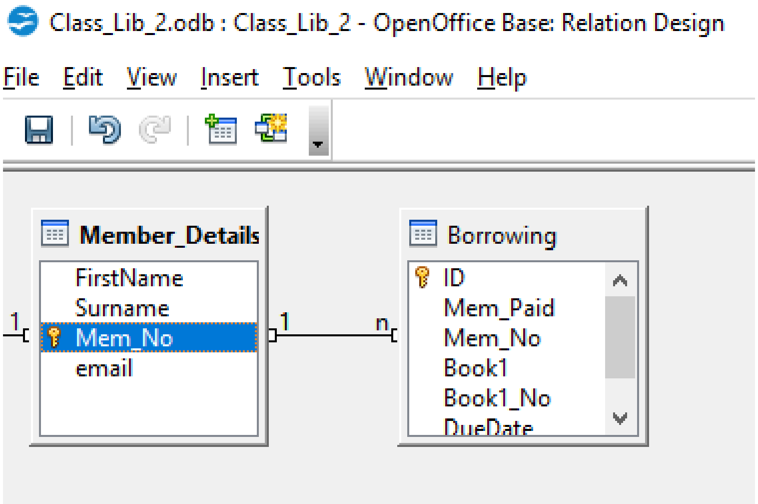
A better solution is to create several tables with links between them. The tables could be created according to function: for example a company may nave a table for *sales details*, *personnel*, *purchase details*, *stock*, *customer details* etc.

Consider our fields in the Members_Borrowings table: some of the fields only change rarely, if at all,  These are fairly static fields, whereas the ones which change fairly frequently are dynamic:

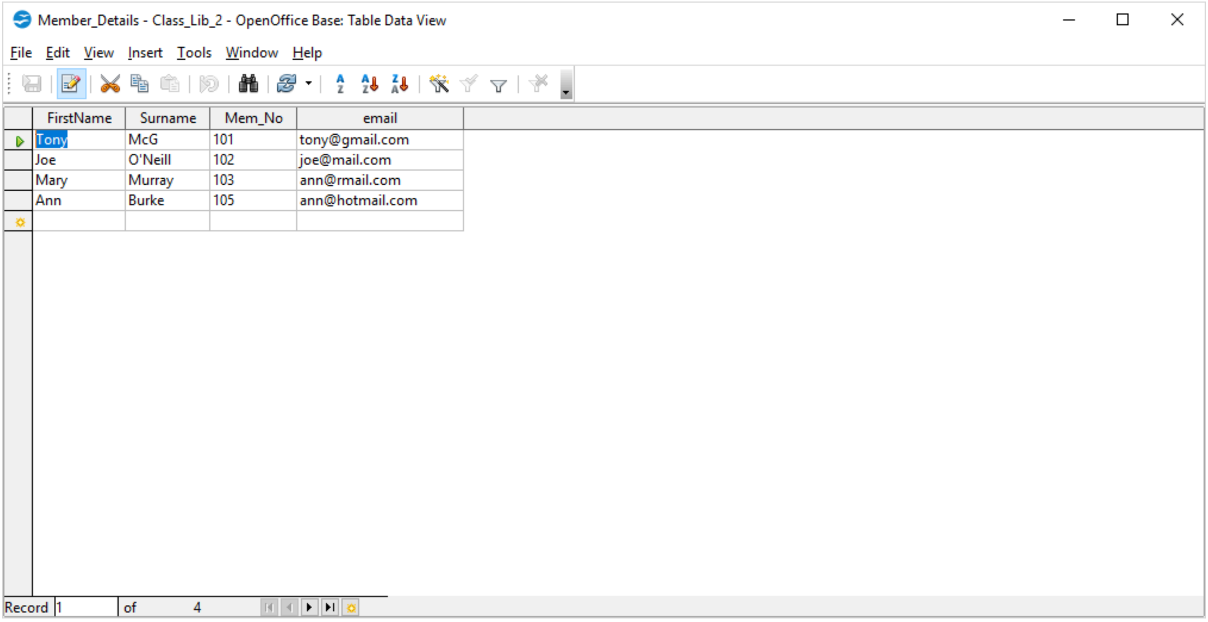
| **Fig 16b:** | **Members_Borrowings table – Categorising Fields** | | | |
|---|---|---|---|---|
| **Personal Details** (quite static!) | FirstName | Surname | Mem_No | Email |
| **Borrowings** (fairly dynamic!) | Book1 | Book1_No | Due Date | Book2 |

We could alter our Class Library design by including two rather than one table.  Here I called the firet table 'Member Details and the second one 'Borrowing' and the fields are split between the two as in Fig 3.  Note that splitting the data into two tables makes the solution more efficient.  Consider if more fields were added to a Tennis Club database, the data may require scrolling across several screen widths just to see one record.

The relationship between the two fields was made at Mem_No id the primary key of Member Details and a foreign key in Borrowing (Fig 17):



The two tables actually look line this (**Fig 18**):

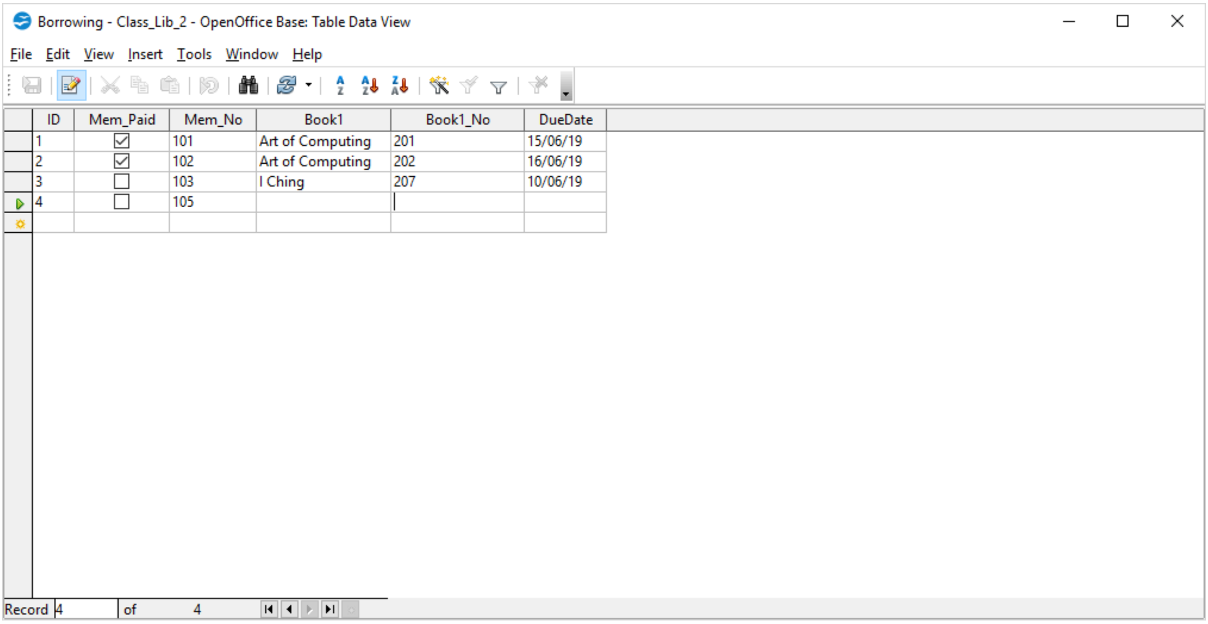


Member_Details - Class_Lib_2 - OpenOffice Base: Table Data View

| FirstName | Surname | Mem_No | email |
|---|---|---|---|
| Tony | McG | 101 | tony@gmail.com |
| Joe | O'Neill | 102 | joe@mail.com |
| Mary | Murray | 103 | ann@rmail.com |
| Ann | Burke | 105 | ann@hotmail.com |

Record 1 of 4

And (**Fig 19**):

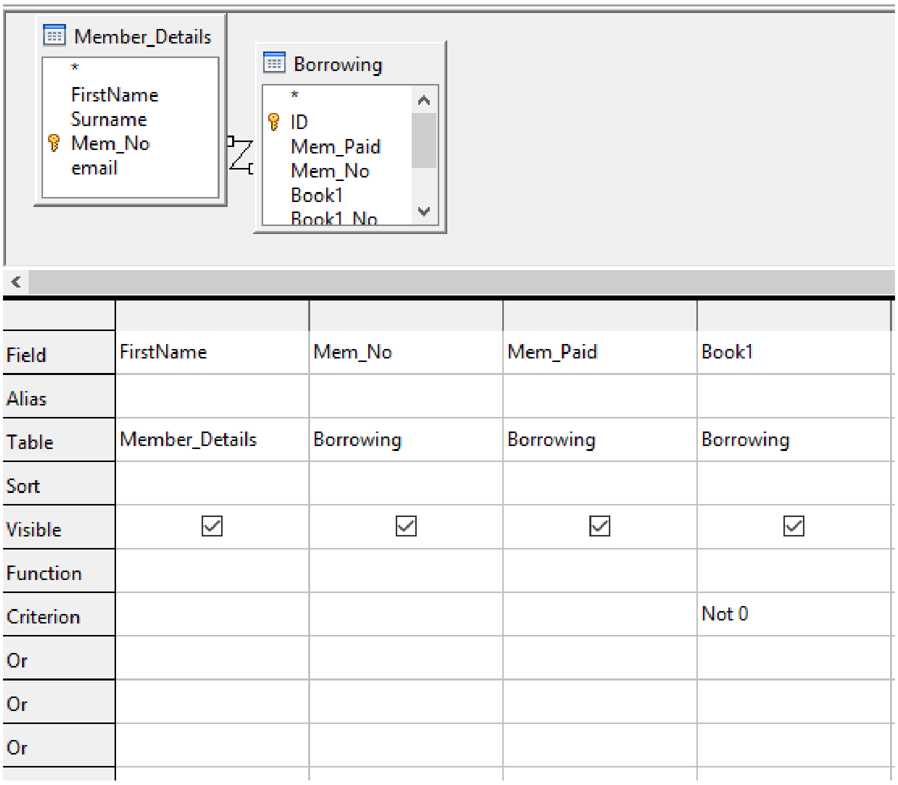


Borrowing - Class_Lib_2 - OpenOffice Base: Table Data View

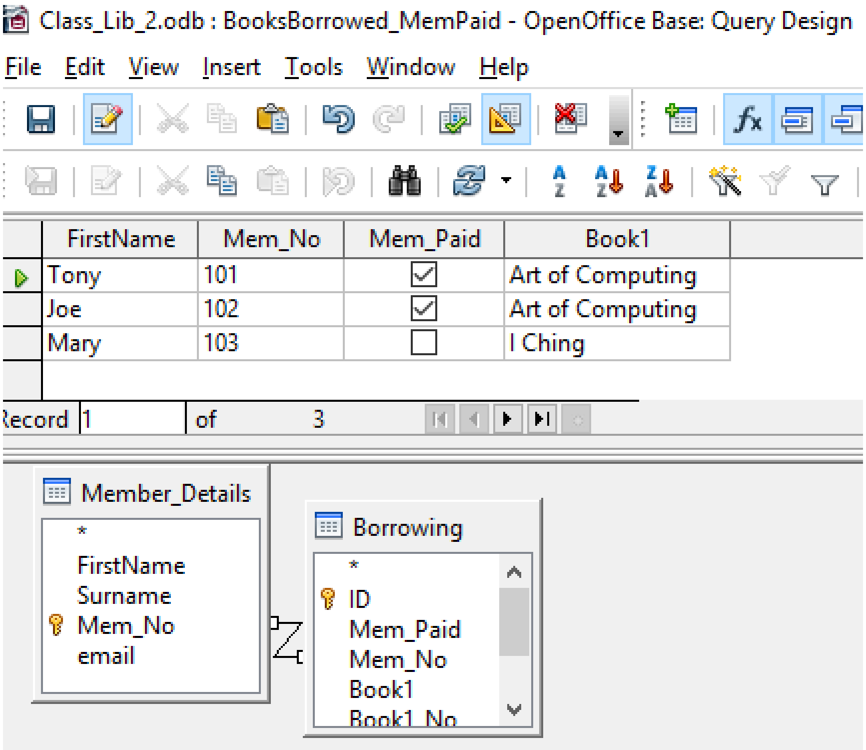
| ID | Mem_Paid | Mem_No | Book1 | Book1_No | DueDate |
|---|---|---|---|---|---|
| 1 | ☑ | 101 | Art of Computing | 201 | 15/06/19 |
| 2 | ☑ | 102 | Art of Computing | 202 | 16/06/19 |
| 3 | ☐ | 103 | I Ching | 207 | 10/06/19 |
| 4 | ☐ | 105 | | | |

Record 4 of 4

A query was run using the two tables on details of those who had a book out, and whether they paid their membership fees or not.  The query design was like this (**Fig 20**):




And it produced results (**Fig 21**):



## Exercise 3: Adding Data and Running Data on the Relational Database ClassLibrary2:

1. Add four more records to Member_Details by direct entry.
2. Insert data for borrowings into the Borrowing table.
3. Design a query which gives the First Name, Surname, Membership No of those who have taken out a book.

## **Exercise 4 : Try creating a one-table or multi-table databse for the school Tennis Club:**

There are some examples below of producing a database for a tennis club. Steps which should be taken are:

1. Planning – brainstorm which fields are required for a one-table solution.
2. Add the fields to the table in *OpenOffice Base*.
3. Create some records using forms and entering data directly to the table.
4. Query the tables using different criteria.
5. Attempt a second (multi-table) solution. Consider that the members will have to book courts every week.
6. Establish relationships in your relational database.

1. Planning – brainstorm which fields are required for a one-table solution.

2. Add the fields to the table in *OpenOffice Base*.

3. Create some records using forms and entering data directly to the table.

4. Query the tables using different criteria.

5. Attempt a second (multi-table) solution. Consider that the members will have to book courts every week.

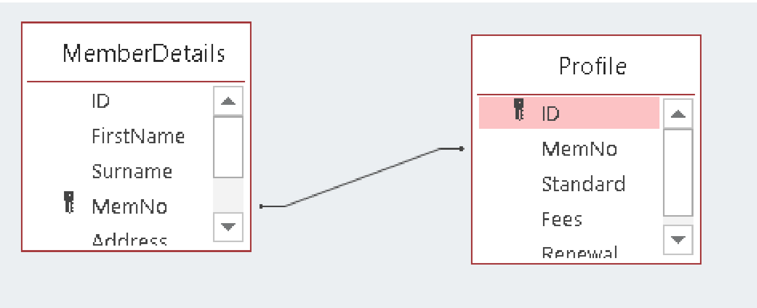
6. Establish relationships in your relational database.

**Example 4: The screenshots below are for reference only and provide part of one solution (Fig 22 a, b, c, d, e):**

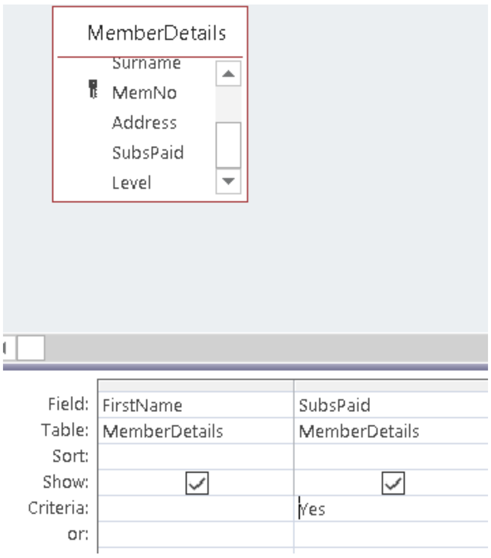
Table:

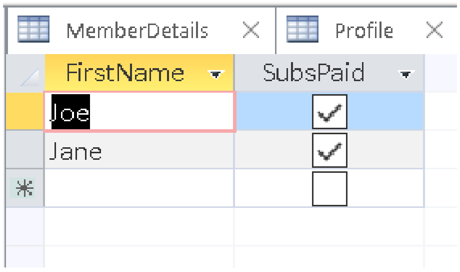
| ID | FirstName | Surname | MemNo | Address | SubsPaid | Level |
|---|---|---|---|---|---|---|
| 1 | Joe | Soap | 121 | 11 Meadovale | ☑ | b |
| 2 | Jane | Doe | 132 | 14 Crescent | ☑ | b |
| 3 | Jimmy | O Dea | 144 | 34 GreenSt | ☐ | a |
| (New) | | | | | ☐ | |

Relationships (Primary key, foreign Key) :




Queries:



Forms:

**MemberDetails**

| | |
|---|---|
| ID | 1 |
| FirstName | Joe |
| Surname | Soap |
| MemNo | 121 |
| Address | 11 Meadovale |

Reports:

**MemberDetails**

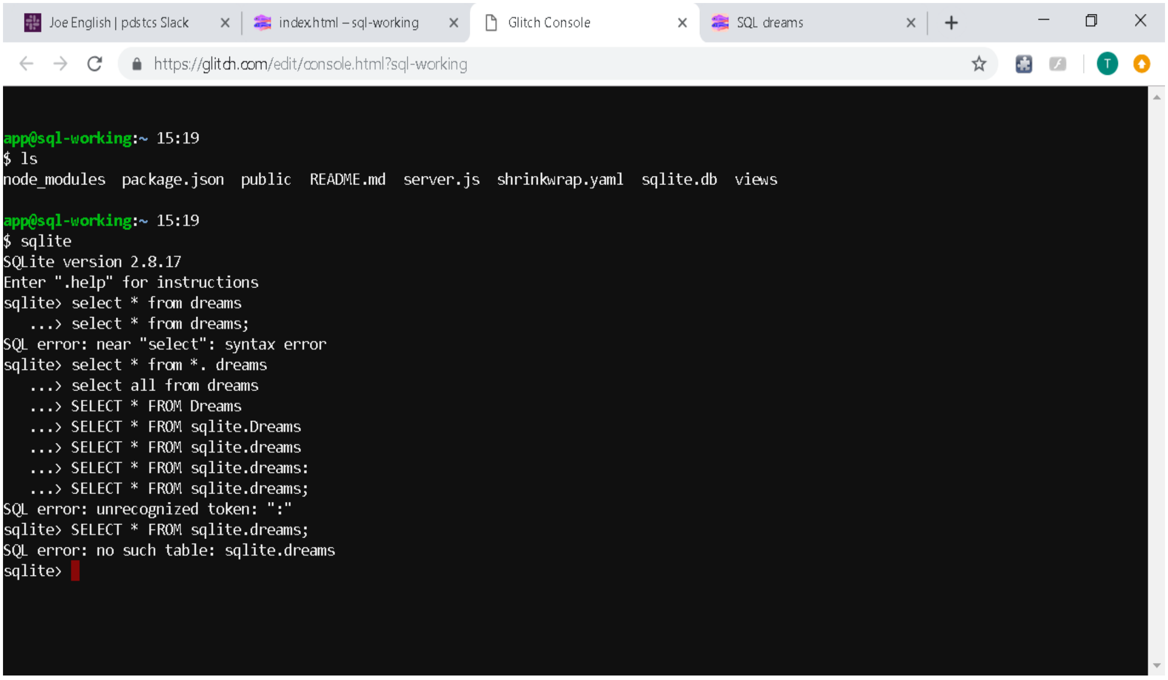
| ID | FirstName | Surname |
|---|---|---|
| 3 | Jimmy | O Dea |
| 1 | Joe | Soap |
| 2 | Jane | Doe |

# SQL Structured Query Language:

SQL is the language of creating databases and their tables; inserting data; querying databases to extract useful information and deleting data and tables.

Some of the DBMSs we will look at use command line, as in *sqlite* below. *MySQL* is also used in some of the examples. Others are integrated into web applications.

Fig 23: Command line *Sqlite* code:




Note:

- The syntax, for example commands end with a semi-colon(;),
- The amount of errors associated with command line slips,
- SQL commands are not case-sensitive.

We'll now show some of the common and most important SQL statements in a *MySQL* database.

These include the basic functions which make up the acronym *CRUD* (Create, Read, Update, Delete). The *CRUD* functions are those required for persistent data storage.

## Example 4: Using MySQL to create databases and tables, read information (Show and Select), update and delete tables.

**Fig 24: Creating a database in MySQL using the command:**

create database *database name*;

```
mysql> create database hodsonbay2;
Query OK, 1 row affected (0.17 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| hodsonbay          |
| hodsonbay2         |
| information_schema |
| mydatabase1        |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| tony2              |
```

Here the database hodsonbay2 is created and the creation was verified using the command "Show databases;" as you can see

**Fig 25: Create a table using the command:**

create table *tablename*(*Column1 Type*, *Column2 Type*…);

```
mysql> use hodsonbay2;
Database changed
mysql> show tables;
Empty set (0.34 sec)

mysql> create table lunchpreferences(id int, FName varchar(255), LName varchar(255));
Query OK, 0 rows affected (1.13 sec)

mysql> show tables;
+----------------------+
| Tables_in_hodsonbay2 |
+----------------------+
| lunchpreferences     |
+----------------------+
1 row in set (0.00 sec)
```

As can be seen in Fig the command “use hodsonbay2;” was used to put the focus on this database. We then made sure that there were no tables there using the “show tables;” command.

The table *lunchpreferences* was created which has Columns (Fields) named “id” , “FName”, “LName”. The “id” field’s type is integer (int) and the other two are character fields.

At this point it is realised that the *lunchpreferences* filed doesn’t actually have a preference. So this is rectified in Fig using the command:

alter table *table_name* add column *column_name column_type* after *existing_column*;

**Fig 26: Adding a Column:**

```
mysql> alter table lunchpreferences add column pref varchar(20) after name;
ERROR 1054 (42S22): Unknown column 'name' in 'lunchpreferences'
mysql> alter table lunchpreferences add column pref varchar(20) after Lname;
Query OK, 0 rows affected (0.50 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show columns;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to
 for the right syntax to use near '' at line 1
mysql> show columns from luncgpreferences;
ERROR 1146 (42S02): Table 'hodsonbay2.luncgpreferences' doesn't exist
mysql> show columns from lunchpreferences;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| id    | int(11)      | YES  |     | NULL    |       |
| FName | varchar(255) | YES  |     | NULL    |       |
| LName | varchar(255) | YES  |     | NULL    |       |
| pref  | varchar(20)  | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
4 rows in set (0.06 sec)
```

Note the errors that occurred on verifying that the column had been added. One was due to not naming the table and the other was a misspelling. With the correct syntax, the columns are there as we required.

We now enter some data using the "Insert" command:

insert into table_name(column1…) values(value1…);

**Fig 27 : Entering data:**

```
mysql> insert into lunchpreferences(id, FName, LName,pref) values (101,Tony, McG,fish);
ERROR 1054 (42S22): Unknown column 'Tony' in 'field list'
mysql> insert into lunchpreferences(id, FName, LName,pref) values (101,'Tony', 'McG','fish');
Query OK, 1 row affected (0.17 sec)

mysql> insert into lunchpreferences(id, FName, LName,pref) values (101,'Tina', 'McH','veg');
Query OK, 1 row affected (0.20 sec)

mysql> insert into lunchpreferences(id, FName, LName,pref) values (101,'Joe', 'Lo','veg');
Query OK, 1 row affected (0.17 sec)

mysql> insert into lunchpreferences(id, FName, LName,pref) values (101,'Jane', 'Fox','chicken');
Query OK, 1 row affected (0.13 sec)
```

You will notice that the id values are the same for the different people mentioned, and this can be corrected using the Update command **(Fig 28):**

update tablename set column1 = value1 where column2 = value2

```
|  101 | Joe   | Lo    | veg     |
|  101 | Jane  | Fox   | chicken |
+------+-------+-------+---------+
4 rows in set (0.00 sec)

mysql> update lunchpreferences set id = 102 where FName = 'Tina';
Query OK, 1 row affected (0.14 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update lunchpreferences set id = 103 where FName = 'Joe';
Query OK, 1 row affected (0.15 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update lunchpreferences set id = 104 where FName = 'Jane';
Query OK, 1 row affected (0.18 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from lunchpreferences;
+------+-------+-------+---------+
| id   | FName | LName | pref    |
+------+-------+-------+---------+
|  101 | Tony  | McG   | fish    |
|  102 | Tina  | McH   | veg     |
|  103 | Joe   | Lo    | veg     |
|  104 | Jane  | Fox   | chicken |
```

So here the unique identifier, id, has been changed for Tina, Joe and Jane.

## Exercise 5: Adding records and verifying

1. Add three more records to the *lunchpreferences* table.
2. Verify the changes have been made using the command:

## Example 4: Querying a table:

As mentioned before, one of the most powerful features of databases is the ability to query. By querying you can get just the records you want with just the fields you want. The next few screenshots show some different queries – one in which you only want certain fields and the other in which you are only interested in records which fulfil certain criteria.

**Fig 29a: Show tables in MySQL**

```
Type 'help;' or '\h' for help. Ty

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydatabase1        |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| tony2              |
| tony7              |
| world              |
+--------------------+
9 rows in set (0.24 sec)

mysql> show tables in tony7;
+-----------------+
| Tables_in_tony7 |
+-----------------+
| table3          |
```

**Fig 29b, c: Select statements:**

```
mysql> select * from tony7.table3;
+-----------+-----------+----------+
| person_id | FirstNAme | City     |
+-----------+-----------+----------+
|        23 | T         | Ramelton |
|        28 | Eef       | Cork     |
|        28 | L         | Down     |
+-----------+-----------+----------+
3 rows in set (0.10 sec)

mysql> select FirstNAme, City from tony7.tables
    -> select FirstNAme, City from tony7.tables;
ERROR 1064 (42000): You have an error in your SQL syntax; ch
near 'select FirstNAme, City from tony7.tables' at line 2
mysql> select FirstNAme, City from tony7.table;
ERROR 1146 (42S02): Table 'tony7.table' doesn't exist
mysql> select FirstNAme, City from tony7.table3;
+-----------+----------+
| FirstNAme | City     |
+-----------+----------+
| T         | Ramelton |
| Eef       | Cork     |
| L         | Down     |
+-----------+----------+
```

```
ysql> select * from tony7.table3 where City = 'Cork' or City ='Ramelton';
-----------+-----------+----------+
 person_id | FirstNAme | City     |
-----------+-----------+----------+
        23 | T         | Ramelton |
        28 | Eef       | Cork     |
-----------+-----------+----------+
 rows in set (0.00 sec)
```

## Exercise 6: Running queries:

1. Run a query on the lunchpreferences table which shows only the people who would like the vegetarian option.
2. Run a query which shows the id and first name only for those whose id is greater than 102.
3. Using the Select Count(*) from lunchupdates, find the number of people who would like the vegetarian option (hint: “where pref = ‘veg’).

## Example 5: Deleting database

If we look at our databases, we notice one called hodsonbay, in addition to the one we were using, hodsonbay2 (**Fig 30**):

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| hodsonbay          |
| hodsonbay2         |
| information_schema |
| mydatabase1        |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| tony2              |
| tony7              |
| tony8              |
| world              |
+--------------------+
12 rows in set (0.04 sec)
```

The hodsonbay database was actually set up in error and can be deleted using the command **(Fig 31):**

Drop database *database_name*;

```
mysql> show databases
    -> show databases;
ERROR 1064 (42000): You have an error in your SQL syntax; check
 for the right syntax to use near 'show databases' at line 2
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| hodsonbay2         |
| information_schema |
| mydatabase1        |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| tony2              |
| tony7              |
| tony8              |
| world              |
+--------------------+
11 rows in set (0.00 sec)
```

The command executed as required as can be seen from the second drop databases command.

**Exercise 7: Use the syntax (*MySQL* command line syntax above and www.w3schools.com/sql) to carry out the CRUD functions.**

1. Create a database (Tennis Club / Class Library example above)
2. Create and populate a table (Tennis club / Class Library members)
3. Insert records
4. Use SQL Select; Where;
5. And, Or, Not
6. SQL Update
7. SQL Delete
8. SQL Min, Max
9. SQL Count, Avg, Sum.

# Databases: Using Web Technologies

**Solution 1**: Uploading to Web Server and Accessing

**Solution 2**: Creating a web page in Base to link to database

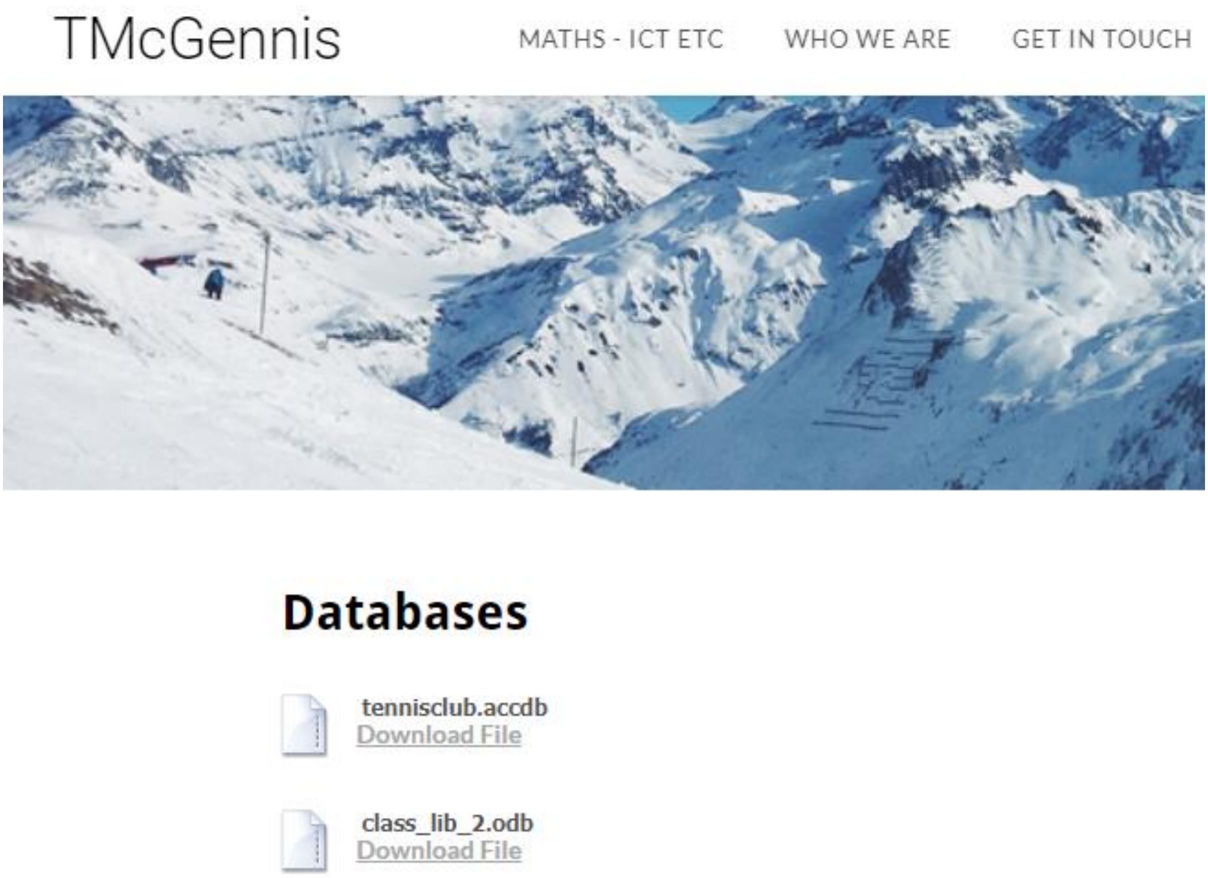
**Solution 3**: Using an interactive web front-end to manipulate a database

These three get progressively more sophisticated, as can be seen if you imagine a solution for a ‘Wedding Present List’.

1. **Uploading to Web Server and Accessing db:**

See the webpage (Fig 32 ) at https://denartha.weebly.com/cs

**Fig 32:**

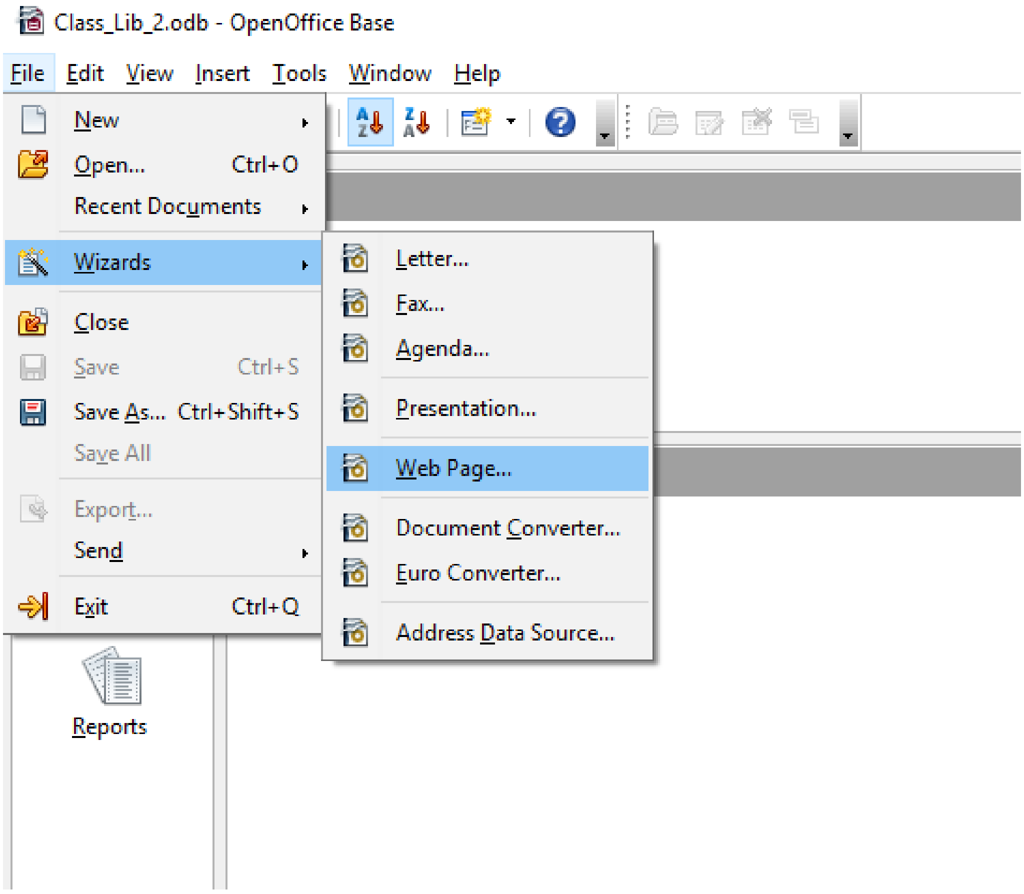



A user can download one of the databases to their own machine, but would need admin rights to upload the updated version. In the ‘Wedding List’ database, this would be inefficient as guests could not update the database with their chosen gift.

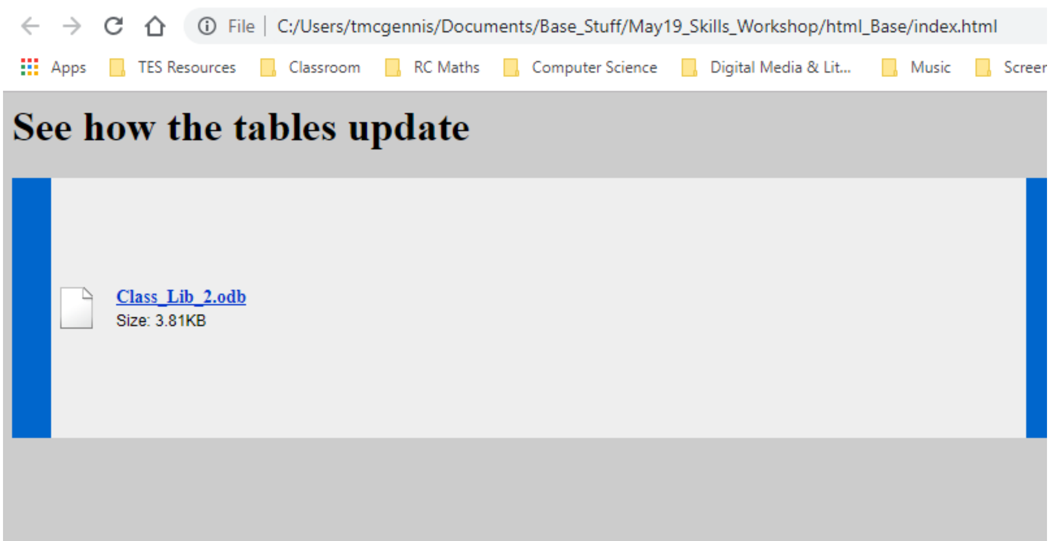
## 2. Creating a Web page in Base to link to Database:

Here we use the Wizard in Base (**Fig 33** ):

(File -> Wizards -> Web page)

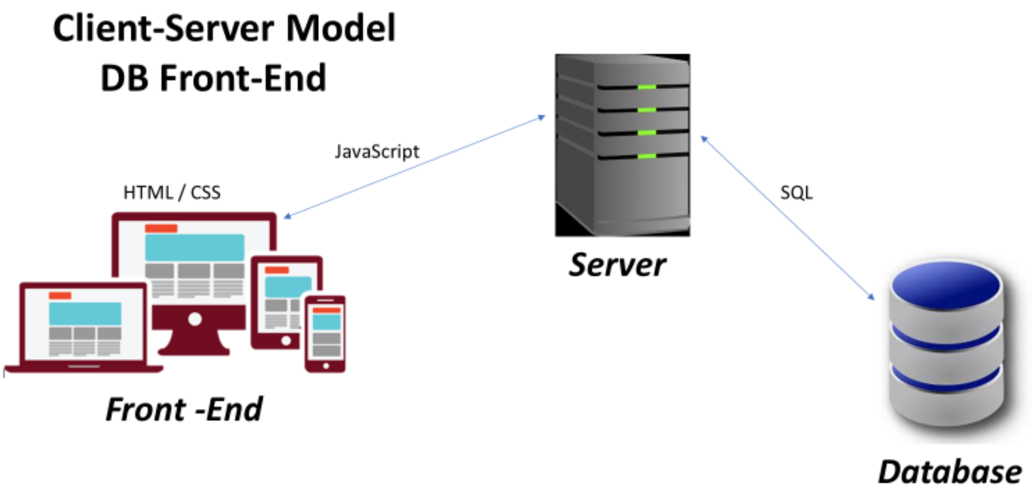



The Wizard creates a web page which can be edited with html (**Fig 34**):

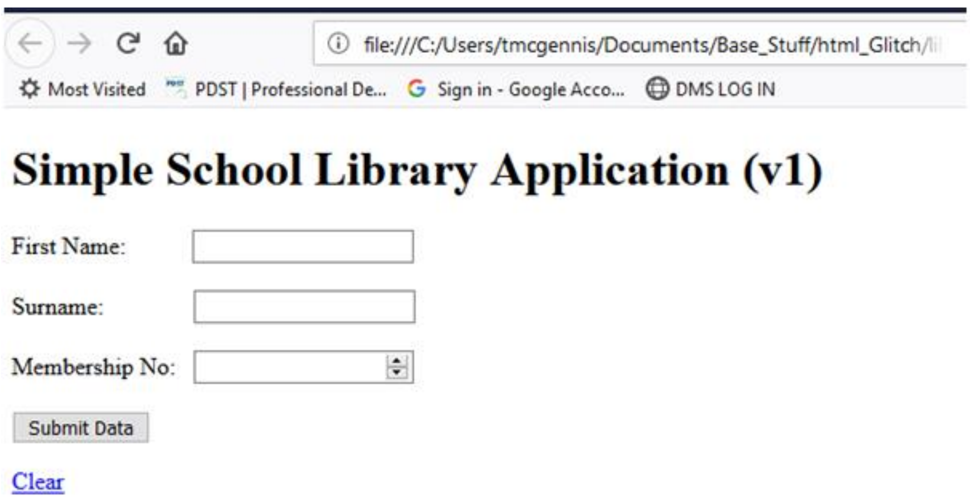



Here the user would need to access the local machine to view or alter the website.

**3. Using an interactive web front-end to manipulate a database (Fig 35):**

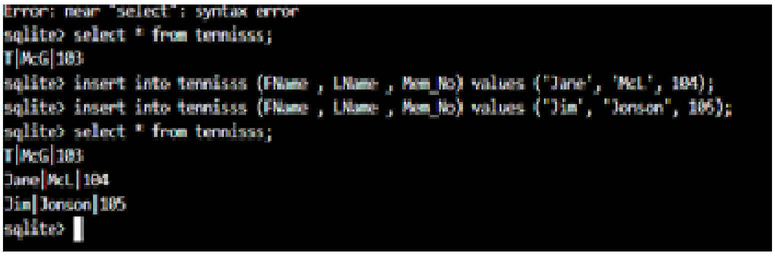



Using HTML / JavaScript / SQL  an interactive solution can be created, whose front-end may look like this (**Fig 36**):



Here is some command line input (INSERT and SELECT statements) in *Sqlite*

*(***Fig 37***)*:



```
Error: near "select": syntax error
sqlite> select * from tennisss;
T|McG|103
sqlite> insert into tennisss (FName , LName , Mem_No) values ('Jane', 'McL', 104);
sqlite> insert into tennisss (FName , LName , Mem_No) values ("Jim", "Jonson", 105);
sqlite> select * from tennisss;
T|McG|103
Jane|McL|104
Jim|Jonson|105
sqlite>
```

## Exercise 7: Database with HTML front-end:

1. Enter some records through the web page in Solution 3.
2. Enter more records through the Console.
3. Query the database through the Console.
4. Query the database through the web page.
5. Discuss the advantages of Solution 3 over the other two.

***We will build on the knowledge and skills gained by completing these exercises to develop a full-stack web application in our final break-out session ***

# Appendices

1. Databases: Websites / Books Consulted:
   - Dr Mikes: http://www.dr-mikes-maths.com/database-glossary.html
   - OpenOffice 'Getting Started Guide': https//www.openoffice.org/documentation/manuals/userguide3
   - 'Computer Science 5th Ed', CS French, Letts, Ch 3
   - 'A-Level Computer Science for AQA Unit 2, Kevin R Bond, Education Computing Service, Ch 10.1 – 10.5
   - Database-driven website: https://www.quackit.com/database/tutorial/database_driven_website.cfm
   - On-line web-database management tool: http://www.glitch.com

2. SQL: Websites / Books Consulted:
   - CodeAcademy: https://www.codecademy.com/articles/sql-commands
   - W3 Schools: https://www.w3schools.com/sql/
   - Wikipedia: https://en.wikipedia.org/wiki/SQL
   - dofactory: https://www.dofactory.com/sql/tutorial
   - Tutorials Point: https://www.tutorialspoint.com/sql/

3. Glossary of database terms (Adapted from Livewire.com):

**ACID**

The ACID model of database design enforces data integrity through:

- **Atomicity**: Each database transaction must follow an all-or-nothing rule, meaning that if any part of the transaction fails, the entire transaction fails.
- **Consistency**: Each database transaction must follow all the database's defined rules; any transaction that would violate these rules is not allowed.
- **Isolation**: Each database transaction will occur independently of any other transaction. For example, if multiple transactions are submitted concurrently, the database will prevent any interference between them.
- **Durability**: Each database transaction will permanently exist in any database failure, via backups or other means.

**Attribute**

A database attribute is a characteristic of a database entity. Simply put, an attribute is a column in a database table, which itself is known as an entity.

**Authentication**

Databases use authentication to ensure that only authorized users can access the database or certain aspects of the database. For example, administrators might be authorized to insert or edit data, while regular employees might be able to only view data. Authentication is implemented with usernames and passwords.

**BASE Model**

The BASE model has been developed as an alternative to the ACID model to serve the needs of (mainly) noSQL databases in which the data is not structured in the same way as RDB required by relational databases. Its primary tenets are:

- **Basic Availability**: The database is available and operational, backed sometimes by data replication distributed across several servers.
- **Soft State**: Countering the ACID model of strict consistency, this tenet states that data does not always have to be consistent and that any enforced consistency is the responsibility of the individual database or developer.
- **Eventual Consistency**: At some undefined future point, the database will achieve consistency.

**Constraints**

A database constraint is a set of rules that define valid data. Multiple types of constraints exist. The primary constraints are:

- **Unique constraints**: A field must contain a unique value in the table.
- **CHECK constraints**: A field can contain only specific data types and even certain allowable values.
- **DEFAULT constraints**: A field will contain a default value if it has no existing value; this eliminates a NULL value.
- **PRIMARY KEY Constraints**: The primary key must be unique.
- **FOREIGN KEY Constraints**: The foreign key must match an existing primary key in another table.

**Database Management System(DBMS)**

DBMS is the software that manages all aspects of working with a database, from storing and securing the data to enforcing data integrity rules, to providing forms for data entry and manipulation. A Relational Database Management System (RDBMS) implements the relational model of tables and relationships between them.

**Entity**

An entity is simply a table in a database. It is described using an Entity-Relationship Diagram, which is a type of graphic that shows the relationships between database tables.

**Functional Dependency**

A functional dependency constraint helps to ensure data validity, and exists when one attribute determines the value of another, described as **A -> B** which means that the value of A determines the value of B, or that B is "functionally dependent" on A. For example, a table in a university that includes records of all students might have a functional dependency between the student ID and the student name, i.e. the unique student ID will determine the value of the name.

**Index**

An index is a data structure that helps speed database queries for large datasets. Database developers create an index on particular columns in a table. The index holds the column values but just pointers to the data in the rest of the table and can be searched efficiently and quickly.

**Key**

A key is a database field whose purpose is to uniquely identify a record. Keys help enforce data integrity and avoid duplication. The main types of keys used in a database are:

- **Candidate keys**: The set of columns that can each uniquely identify a record and from which the primary key is chosen.
- **Primary keys**: The key chosen to uniquely identify a record in a table. This key cannot be NULL.
- **Foreign keys**: The key linking a record to a record in another table. A table's foreign key must exist as the primary key of another table.

**Normalization**

To normalize a database is to design its tables (relations) and columns (attributes) in a way to ensure data integrity and to avoid duplication. The primary levels of normalization are First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), and Boyce-Codd Normal Form (BCNF).

**NoSQL**

NoSQL is a database model developed to respond to the need for storing unstructured data such as emails, social media posts, video, or images. Rather than using SQL and the strict ACID model to ensure data integrity, NoSQL follows the less-strict BASE model. A NoSQL database schema does not use tables to store data; rather, it might use a key/value design or graphs.

**Null**

The value NULL is frequently confused to mean "none" or zero; however, it actually means "unknown." If a field has a value of NULL, it is a placeholder for an unknown value. Structured Query Language (SQL) uses the IsNull and IsNot Null tests.

**Query**

A database query is how users interact with a database. It is usually written in SQL and can be either a *select* query or an *action* query. A select query requests data from a database; an action query changes, updates, or adds data. Some databases provide forms that hide the semantics of the query, allowing users to easily request information without having to understand SQL.

**Schema**

A database schema is the design of tables, columns, relations, and constraints that make up a database. Schemas are usually described using the SQL CREATE statement.

**Stored Procedure**

A stored procedure is a pre-compiled query or SQL statement that is used in a RDBMS.

**Structured Query Language**

Structured Query Language, or SQL, is the most commonly used language to access data from a database. The Data Manipulation Language (DML) contains the subset of SQL commands used most frequently and includes SELECT, INSERT, UPDATE and DELETE.

**Trigger**

A trigger is a stored procedure set to execute given a particular event, usually a change to a table's data. For example, a trigger might be designed to write to a log, gather statistics, or compute a value.

**View**

A database view is a filtered set of data displayed to the end user in order to hide data complexity and streamline the user experience. A view can join data from two or more tables and contains a subset of information.

4. Glossary of SQL Commands:
(Source : https://www.w3schools.com/sql/sql_quickref.asp)

| SQL Statement | Syntax |
|---|---|
| AND / OR | SELECT column_name(s)<br>FROM table_name<br>WHERE condition<br>AND\|OR condition |
| ALTER TABLE | ALTER TABLE table_name<br>ADD column_name datatype<br><br>Or<br><br>ALTER TABLE table_name<br>DROP COLUMN column_name |
| AS (alias) | SELECT column_name AS column_alias<br>FROM table_name<br><br>or<br><br>SELECT column_name<br>FROM table_name  AS table_alias |
| BETWEEN | SELECT column_name(s)<br>FROM table_name<br>WHERE column_name<br>BETWEEN value1 AND value2 |
| CREATE DATABASE | CREATE DATABASE database_name |
| CREATE TABLE | CREATE TABLE table_name<br>(<br>column_name1 data_type,<br>column_name2 data_type,<br>column_name3 data_type,<br>...<br>) |
| CREATE INDEX | CREATE INDEX index_name<br>ON table_name (column_name)<br><br>or<br><br>CREATE UNIQUE INDEX index_name<br>ON table_name (column_name) |

| | |
|---|---|
| CREATE VIEW | CREATE VIEW view_name AS<br>SELECT column_name(s)<br>FROM table_name<br>WHERE condition |
| DELETE | DELETE FROM table_name<br>WHERE some_column=some_value<br><br>or<br><br>DELETE FROM table_name<br>(**Note:** Deletes the entire table!!)<br><br>DELETE * FROM table_name<br>(**Note:** Deletes the entire table!!) |
| DROP DATABASE | DROP DATABASE database_name |
| DROP INDEX | DROP INDEX table_name.index_name (SQL Server)<br>DROP INDEX index_name ON table_name (MS Access)<br>DROP INDEX index_name (DB2/Oracle)<br>ALTER TABLE table_name<br>DROP INDEX index_name (MySQL) |
| DROP TABLE | DROP TABLE table_name |
| EXISTS | IF EXISTS (SELECT * FROM table_name WHERE id = ?)<br>BEGIN<br>--do what needs to be done if exists<br>END<br>ELSE<br>BEGIN<br>--do what needs to be done if not<br>END |
| GROUP BY | SELECT column_name, aggregate_function(column_name)<br>FROM table_name<br>WHERE column_name operator value<br>GROUP BY column_name |
| HAVING | SELECT column_name, aggregate_function(column_name)<br>FROM table_name<br>WHERE column_name operator value<br>GROUP BY column_name<br>HAVING aggregate_function(column_name) operator value |
| IN | SELECT column_name(s)<br>FROM table_name<br>WHERE column_name<br>IN (value1,value2,..) |
| | |

| | |
|---|---|
| INSERT INTO | INSERT INTO table_name<br>VALUES (value1, value2, value3,....)<br><br>*or*<br><br>INSERT INTO table_name<br>(column1, column2, column3,...)<br>VALUES (value1, value2, value3,....) |
| FULL JOIN | SELECT column_name(s)<br>FROM table_name1<br>FULL JOIN table_name2<br>ON table_name1.column_name=table_name2.column_name |
| LIKE | SELECT column_name(s)<br>FROM table_name<br>WHERE column_name LIKE pattern |
| ORDER BY | SELECT column_name(s)<br>FROM table_name<br>ORDER BY column_name [ASC\|DESC] |
| SELECT | SELECT column_name(s)<br>FROM table_name |
| SELECT * | SELECT *<br>FROM table_name |
| SELECT DISTINCT | SELECT DISTINCT column_name(s)<br>FROM table_name |
| SELECT INTO | SELECT *<br>INTO new_table_name [IN externaldatabase]<br>FROM old_table_name<br><br>*or*<br><br>SELECT column_name(s)<br>INTO new_table_name [IN externaldatabase]<br>FROM old_table_name |
| SELECT TOP | SELECT TOP number\|percent column_name(s)<br>FROM table_name |
| TRUNCATE TABLE | TRUNCATE TABLE table_name |
| UNION | SELECT column_name(s) FROM table_name1<br>UNION<br>SELECT column_name(s) FROM table_name2 |
| UNION ALL | SELECT column_name(s) FROM table_name1<br>UNION ALL<br>SELECT column_name(s) FROM table_name2 |

| | |
|---|---|
| UPDATE | UPDATE table_name<br>SET column1=value, column2=value,...<br>WHERE some_column=some_value |
| WHERE | SELECT column_name(s)<br>FROM table_name<br>WHERE column_name operator value |