Overview
The week 3 practical required us to connect to our MySQL database on the school host servers. Once connected to our own database we were required to write a program using JDBC, to read a given data file into a MySQL database and to perform certain queries over the data. To connect to the database we were required to read the MySQL connection parameters from database.properties and then connect to the MySQL database on the school host server using the connection properties. Once connected to the database the program should check whether a table for the data already exists in the database, and if so, delete it and then create a new table. The program was then required to read the data from a given input file and insert it into the table.

Design
I first attempted this practical using JDBC, I then decided to move on to attempt it using ORM and finally once I had the ORM working I implemented a relational database with multiple tables using ORM. I will first explain the design of my JDBC program.
The design of my project for reading data in from a file bears, close resemblance to my previous practical, due the simplicity and efficiency of the structure in my previous did not feel the need to write a new framework.

JDBC
First the a new properties object is created and the key value pairs from the data.proerties files is loaded in. Once the appropriate properties are loaded in the program then attempts to connect to the database. If a successful connection is made, a connection object is returned. Upon successfully connecting the program checks using an SQL query whether a table for with the same name for the new table exists in the database, and if so, it is deleted and a new table is then created. The program then moves on and starts to read the data from the CSV file. A new instance of the experiment class is initiated and then a while loop is then entered; the terminating condition is met when the line of the text file is null e.g. end of file. When the while loop is entered the function from the ExperimentReader class is called, 'hasNextExperiment. This function returns a Boolean value, which is true when the end of the file is met, otherwise the returned value will be false. Within this function another function is called, findNextnExperiment. The find findNextnExperiment function creates a new instance of the Experiment class, which is a sub class (nested class) of the ExperimentReader class. The Experiment class contains attributes for each column in the CSV file. The first line of the CSV file is ignored and the subsequent lines are then read in and split up by commas. Each value that has been split in the current row read in, is then added to an array. The values in the array are then assighned to the attributes of the nextExperiment object. A new nextExperiment object is created for every row in the CSV file. After the row has been read in and the values are assigned to the nextExperiment object, the program then adds the data held in the object to the database by using using using a prepared statement which takes the values stored in the nextExperiment object attributes and adds them to the statement in the correct position. This process continues until the end of the CSV file is reached. The class 'DatabaseMapipulation' is used to query the database created to find statics from the data. I have used the prepared statement object in order to query the the database, in order to prevent any SQL injection.

ORM
I have used the example on studres in order to get the frame work for the ORM program. However, I have significantly changed the source code, the main changes include:

- There is now only one main method
- I have implemented a user interface
- The properties for the database are now read in from a .properties file.
- Multiple tables and relations between the tables are made with primary and foreign keys

I have included an ER diagram, which shows the tables I decided to create, the entities for each table and the relationships between them.

The ORM program works in a similar way as the JDBC program does with regard to reading in data from the CSV file and the way in which the database.properties file is read in and the connection properties are set up. However, because I am using multiple tables I decided to create multiple classes for each table which are now longer nested and have appropriate attributes for the table column names. In order to make sure my database was normalised I ensured the new database design contained no repeating attributes or groups of attributes and that there were no partial key dependencies. In order to do this, each line in the database new

Experiment, Spectrometer and User objects are created. These classes have primary keys, which are annotated as foreign keys in the Samples class. The Entitymanger.find function is then used to find previous entry's in the database with mating key values as those read in from the CSV file. If there are no matches found the program uses the values read in from the CSV file to initialise the new object and the object is added to the database. In the main class there are function which are used through the program, the first 2 functions are 'getInputInt' and 'getInputString' which return and integer and string respectively. The next functions allow the program to find rows in the tables and return an object correlating to that row. The program will not however return an object if an invalid input is entered, in this event it will simply prompt the user to input something valid. Next in the 'Main' class is the main method here the EntityManager is created and is used through the program. The main method sets up the connections and then calls the 'storeObjects' method in the Store class. This method takes in the CSV file and like the JDBC program uses a while loop to create appropriate objects until the end of the CSV file is met. I have created a simple interface which is coded in the main method, this interface simply allows the user to choose what they which to do next, e.g. print statics, update or delete a row etc. The 'Remove' and 'Retrieve' classes allows uses to remove or retrieve, respectively, a sample from the database by inputting the sample ID to the function in the main method. The 'RetreiveAll' class retrieves all the samples in the database by using a querry. The 'RetrieveStats' class has the following methods 'findSamplesOnSpec' , 'findTimes' and 'findNoOfExperiment'. The 'findSamplesOnSpec' and 'findNoOfExperiment' allow the user to input a spectrometer ID or an Experiment ID, and by using a query will return all the experiments carried out on that spectrometer or the number of experiments carried out with that substance. The 'findTimes' method uses more complex quires to find the longest, shortest, total and average time. In order to find the total and average time I had to first make the query convert the rows in the duration column to seconds and then use the SUM and AVG functions. The final class in the ORM program was the update class which a user of the program to update values in the database. This class proved much harder to program due to the complexity and issues surrounding the format of data input. When the user chooses to update something in the database, they must first input the sample ID. The first method in this class is called 'getGetters' and is used to get all the **public** getters in the entity's classes, all primary key getters where made private. The function takes in a class as parameter and uses the .getMethods function in a for loop to find all the fucntions in the input class. All public getter functions in the input class are then output to the console, excluding the 'ClassGetter' functions. This displays to the user what they are able to update. The user then can choose which attribute from the entity classes they wish to change, which in turn will update the data in the database. When users wish to update the 'Record_submit_time' or 'Sample_duration' the program will enforce format validation. This is because in the database the Record_submit_time' and 'Sample_duration' columns are of type 'DATETIME' and 'TIME' and thus if data is not in the valid format that the database recognises there will be an error.

Extensions

I've implemented the following extensions:

- ORM and JDBC programs
- Storing the data in multiple tables with foreign and primary keys
- Additional statistics – average time
- Ability to print data from all spectrometers
- Ability to calculate the number of samples run with all the different experiments
- The ability to retrieve specific records from the database
- The ability to remove and update data in the database
- A basic user interface
- An ER diagram
- A work around to not 'throttle' the servers

Research

My ORM solution allows me to use the 'data-large.csv' file without throttling the server. I have done this by using a different method for adding the data to the database. Rather than adding the data by using the 'INSERT' query and adding one line at a time, the JDBC program now has the option to load the file into the database instead. I achieved this by altering the query from 'INSERT' to LOAD DATA LOCAL INFILE.

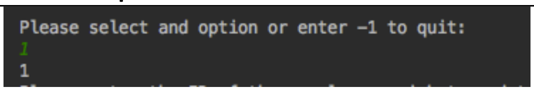
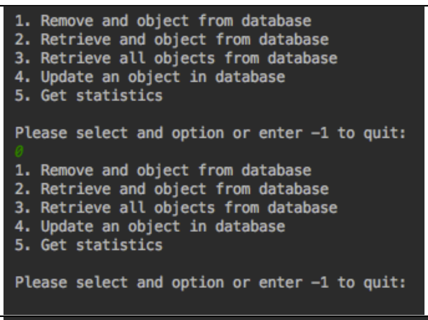
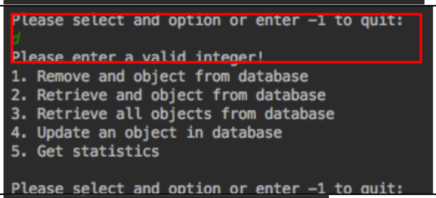
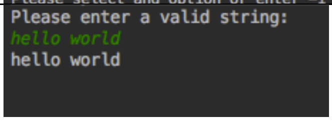
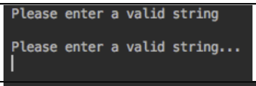
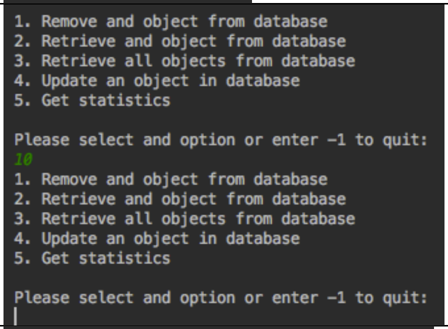
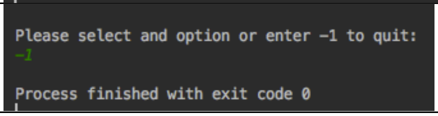
```
java.sql.PreparedStatement statement1 = connection.prepareStatement("LOAD DATA LOCAL INFILE
'data-small.csv'\n" +
        "INTO TABLE Practical3 \n" +
        "FIELDS TERMINATED BY ',' \n" +
        "LINES TERMINATED BY '\\n' " +
        "(sample_id , record_submit_time, sample_holderno, sample_duration, exp_id, " +
        "exp_name, exp_description ,user_id , group_abbr, solvent_abbr, spectrometer_id, " +
        "spectrometer_name, spectrometer_capacity);");
statement1.execute();
```

__Testing__
To test the core functionality of my program I will use the various files supplied as an input and make sure the tables and the values in the table are correct. I will also try and update and romove data from and check if the processes has completed as expected. To carry out erroneous and extreme testing I will do the following tests:

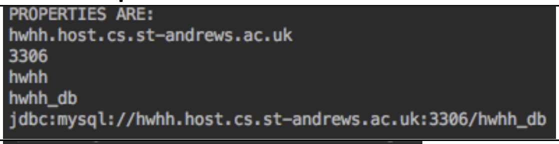
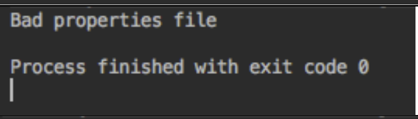
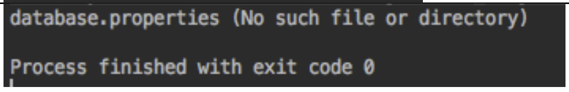
- Try and input an invalid file
- Try and input an invalid number when the user input is required.

Tests for both ORM and JDBC

Invalid inputs:

Test Description	Expected output	Actual output	Comment
Enter a valid integer	The input is accepted and the return from the function is the input number	 Please select and option or enter -1 to quit: 1 1	Works as expected
Enter invalid integer (e.g. 0)	The user is asked to re-enter a number	 1. Remove and object from database 2. Retrieve and object from database 3. Retrieve all objects from database 4. Update an object in database 5. Get statistics Please select and option or enter -1 to quit: 0 1. Remove and object from database 2. Retrieve and object from database 3. Retrieve all objects from database 4. Update an object in database 5. Get statistics Please select and option or enter -1 to quit:	Works as expected
Enter letter instead of integer	The user is asked to re-enter a number	 Please select and option or enter -1 to quit: d Please enter a valid integer! 1. Remove and object from database 2. Retrieve and object from database 3. Retrieve all objects from database 4. Update an object in database 5. Get statistics Please select and option or enter -1 to quit:	Works as expected
Enter a valid string	The input is accepted and the return from the function is the input string	 Please enter a valid string: hello world hello world	Works as expected
Enter nothing	The user is asked to enter a valid string	 Please enter a valid string Please enter a valid string...	Works as expected
Enter an invalid choice	The user is asked to re-enter a number	 1. Remove and object from database 2. Retrieve and object from database 3. Retrieve all objects from database 4. Update an object in database 5. Get statistics Please select and option or enter -1 to quit: 10 1. Remove and object from database 2. Retrieve and object from database 3. Retrieve all objects from database 4. Update an object in database 5. Get statistics Please select and option or enter -1 to quit:	Works as expected
User enter -1 to quit program	The program exits	 Please select and option or enter -1 to quit: -1 Process finished with exit code 0	Works as expected

Properties Test:

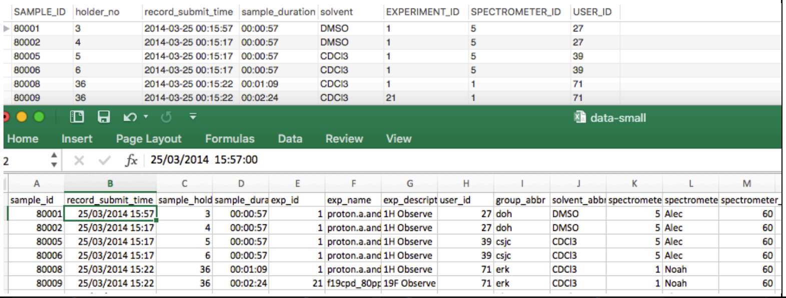
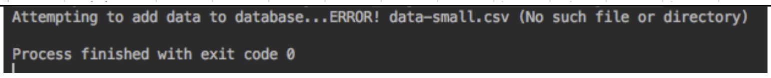
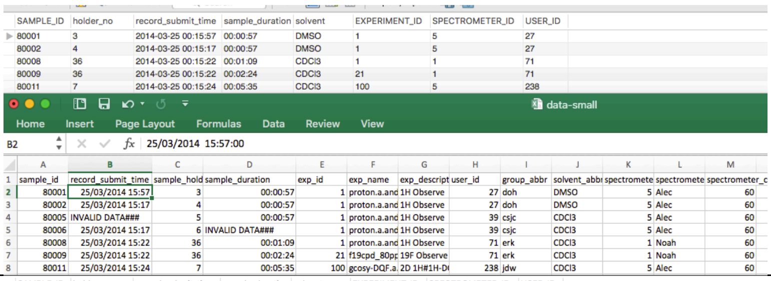
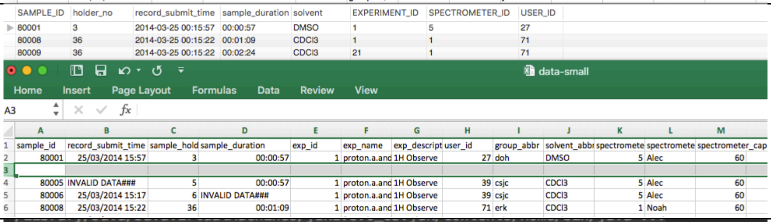
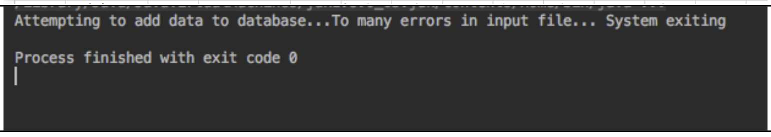
Test Description	**Expected output**	**Actual output**	**Comment**
Properties file valid	Output to console the properties		Works as expected
Properties file has invalid content (username!=helloworld)	Outputs error message and program exits		Works as expected
No properties file	Outputs error message and program exits		Works as expected

```
PROPERTIES ARE:
hwhh.host.cs.st-andrews.ac.uk
3306
hwhh
hwhh_db
jdbc:mysql://hwhh.host.cs.st-andrews.ac.uk:3306/hwhh_db
```

```
Bad properties file

Process finished with exit code 0
```

```
database.properties (No such file or directory)

Process finished with exit code 0
```

Read and Add data test:

Test Description	**Expected output**	**Actual output**	**Comment**
Valid CSV	The data from the CSV file is read in line by line. New objects. Rows are appended to the database with either the object or the objects attributes	The file is read in line by line, all the correct objects are created and the data is saved in the correct format in the database 	Works as expected
No CSV file	Outputs error message and program exits		Works as expected
Data in wrong format	That line of the CSV file is skipped	The line with 'INVALID DATA' in a cell where skipped and not added to the database all valid rows where added 	Works as expected
Missing row	That line of the CSV file is skipped		
Missing column	Outputs error message and program exits		

Valid CSV:

SAMPLE_ID	holder_no	record_submit_time	sample_duration	solvent	EXPERIMENT_ID	SPECTROMETER_ID	USER_ID
80001	3	2014-03-25 00:15:57	00:00:57	DMSO	1	5	27
80002	4	2014-03-25 00:15:17	00:00:57	DMSO	1	5	27
80005	5	2014-03-25 00:15:17	00:00:57	CDCl3	1	5	39
80006	6	2014-03-25 00:15:17	00:00:57	CDCl3	1	5	39
80008	36	2014-03-25 00:15:22	00:01:09	CDCl3	1	1	71
80009	36	2014-03-25 00:15:22	00:02:24	CDCl3	21	1	71

data-small

2 25/03/2014 15:57:00

A	B	C	D	E	F	G	H	I	J	K	L	M
sample_id	record_submit_time	sample_hold	sample_dura	exp_id	exp_name	exp_descript	user_id	group_abbr	solvent_abbı	spectromete	spectromete	spectrometer_
80001	25/03/2014 15:57	3	00:00:57	1	proton.a.and	1H Observe	27	doh	DMSO	5	Alec	60
80002	25/03/2014 15:17	4	00:00:57	1	proton.a.and	1H Observe	27	doh	DMSO	5	Alec	60
80005	25/03/2014 15:17	5	00:00:57	1	proton.a.and	1H Observe	39	csjc	CDCl3	5	Alec	60
80006	25/03/2014 15:17	6	00:00:57	1	proton.a.and	1H Observe	39	csjc	CDCl3	5	Alec	60
80008	25/03/2014 15:22	36	00:01:09	1	proton.a.and	1H Observe	71	erk	CDCl3	1	Noah	60
80009	25/03/2014 15:22	36	00:02:24	21	f19cpd_80pp	19F Observe	71	erk	CDCl3	1	Noah	60

No CSV file:

```
Attempting to add data to database...ERROR! data-small.csv (No such file or directory)

Process finished with exit code 0
```

Data in wrong format:

SAMPLE_ID	holder_no	record_submit_time	sample_duration	solvent	EXPERIMENT_ID	SPECTROMETER_ID	USER_ID
80001	3	2014-03-25 00:15:57	00:00:57	DMSO	1	5	27
80002	4	2014-03-25 00:15:17	00:00:57	DMSO	1	5	27
80008	36	2014-03-25 00:15:22	00:01:09	CDCl3	1	1	71
80009	36	2014-03-25 00:15:22	00:02:24	CDCl3	21	1	71
80011	7	2014-03-25 00:15:24	00:05:35	CDCl3	100	5	238

data-small

B2 25/03/2014 15:57:00

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	sample_id	record_submit_time	sample_hold	sample_duration	exp_id	exp_name	exp_descript	user_id	group_abbr	solvent_abbı	spectromete	spectromete	spectrometer_c
2	80001	25/03/2014 15:57	3	00:00:57	1	proton.a.and	1H Observe	27	doh	DMSO	5	Alec	60
3	80002	25/03/2014 15:17	4	00:00:57	1	proton.a.and	1H Observe	27	doh	DMSO	5	Alec	60
4	80005	INVALID DATA###	5	00:00:57	1	proton.a.and	1H Observe	39	csjc	CDCl3	5	Alec	60
5	80006	25/03/2014 15:17	6	INVALID DATA###	1	proton.a.and	1H Observe	39	csjc	CDCl3	5	Alec	60
6	80008	25/03/2014 15:22	36	00:01:09	1	proton.a.and	1H Observe	71	erk	CDCl3	1	Noah	60
7	80009	25/03/2014 15:22	36	00:02:24	21	f19cpd_80pp	19F Observe	71	erk	CDCl3	1	Noah	60
8	80011	25/03/2014 15:24	7	00:05:35	100	gcosy-DQF.a.	2D 1H#1H-D(	238	jdw	CDCl3	5	Alec	60

Missing row:

SAMPLE_ID	holder_no	record_submit_time	sample_duration	solvent	EXPERIMENT_ID	SPECTROMETER_ID	USER_ID
80001	3	2014-03-25 00:15:57	00:00:57	DMSO	1	5	27
80008	36	2014-03-25 00:15:22	00:01:09	CDCl3	1	1	71
80009	36	2014-03-25 00:15:22	00:02:24	CDCl3	21	1	71

data-small

A3

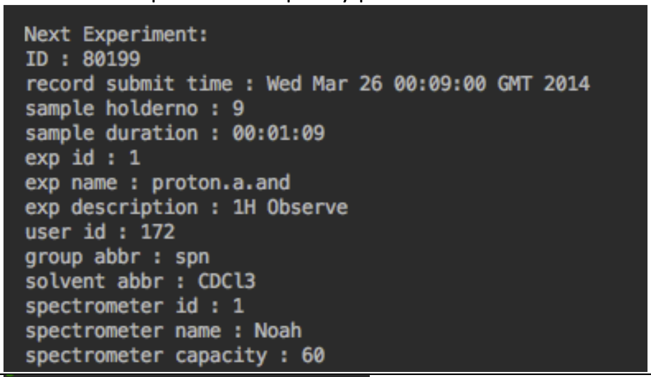
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	sample_id	record_submit_time	sample_hold	sample_duration	exp_id	exp_name	exp_descript	user_id	group_abbr	solvent_abbı	spectromete	spectromete	spectrometer_cap
2	80001	25/03/2014 15:57	3	00:00:57	1	proton.a.and	1H Observe	27	doh	DMSO	5	Alec	60
3													
4	80005	INVALID DATA###	5	00:00:57	1	proton.a.and	1H Observe	39	csjc	CDCl3	5	Alec	60
5	80006	25/03/2014 15:17	6	INVALID DATA###	1	proton.a.and	1H Observe	39	csjc	CDCl3	5	Alec	60
6	80008	25/03/2014 15:22	36	00:01:09	1	proton.a.and	1H Observe	71	erk	CDCl3	1	Noah	60

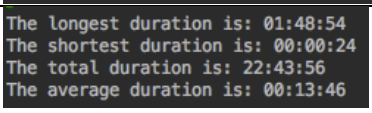
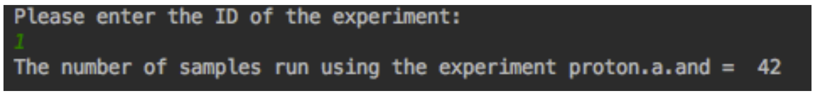
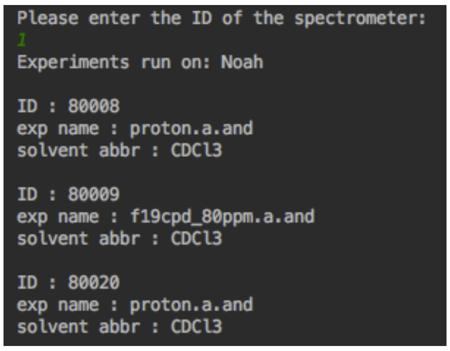
Missing column:

```
Attempting to add data to database...To many errors in input file... System exiting

Process finished with exit code 0
```

Print all records and stats test:

Test Description	**Expected output**	**Actual output**	**Comment**

Print all the records	All the rows are formatted and pinted to the command line	Worked as expected with pretty printed date  `Next Experiment:`
		`ID : 80199`
		`record submit time : Wed Mar 26 00:09:00 GMT 2014`
		`sample holderno : 9`
		`sample duration : 00:01:09`
		`exp id : 1`
		`exp name : proton.a.and`
		`exp description : 1H Observe`
		`user id : 172`
		`group abbr : spn`
		`solvent abbr : CDCl3`
		`spectrometer id : 1`
		`spectrometer name : Noah`
		`spectrometer capacity : 60`
			Works as expected

Print the stats	The statists are correctly calculated and printed to the command line	 `The longest duration is: 01:48:54`
		`The shortest duration is: 00:00:24`
		`The total duration is: 22:43:56`
		`The average duration is: 00:13:46`
		 `Please enter the ID of the experiment:`
		`1`
		`The number of samples run using the experiment proton.a.and =  42`
		 `Please enter the ID of the spectrometer:`
		`1`
		`Experiments run on: Noah`
		
		`ID : 80008`
		`exp name : proton.a.and`
		`solvent abbr : CDCl3`
		
		`ID : 80009`
		`exp name : f19cpd_80ppm.a.and`
		`solvent abbr : CDCl3`
		
		`ID : 80020`
		`exp name : proton.a.and`
		`solvent abbr : CDCl3`
			Works as expected

JDBC Tests:

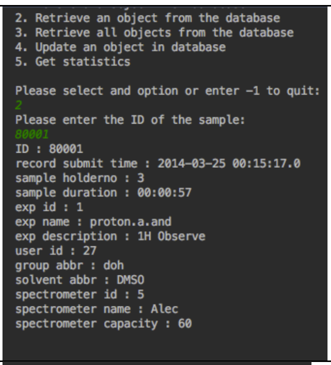
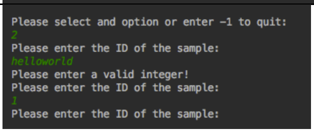
Throttling test:

Test Description	**Expected output**	**Actual output**	**Comment**
Attempt to exceed the normal request	The program should be load in the large data file without exceeding the maximum number of /queries/updates per hour	The program was successfully able to load in the large data file.	Works as expected

ORM

Retrieve Sample:

Test Description	**Expected output**	**Actual output**	**Comment**

Search for valid Sample	The program will output to the console the sample information		Works as expected
Search for in-valid Sample	The program will prompt the user to re-enter the sample ID		Works as expected

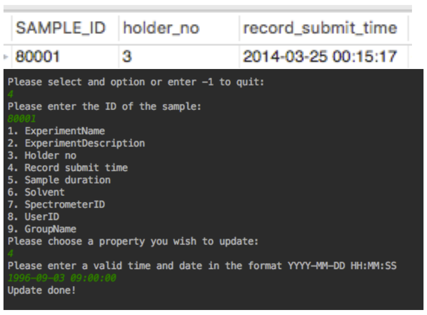
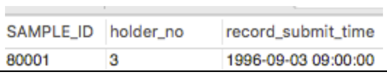
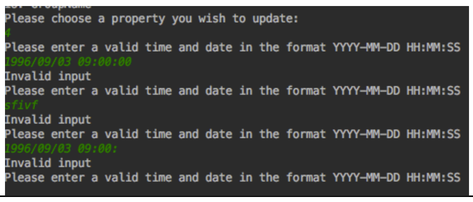
```
2. Retrieve an object from the database
3. Retrieve all objects from the database
4. Update an object in database
5. Get statistics

Please select and option or enter -1 to quit:
2
Please enter the ID of the sample:
80001
ID : 80001
record submit time : 2014-03-25 00:15:17.0
sample holderno : 3
sample duration : 00:00:57
exp id : 1
exp name : proton.a.and
exp description : 1H Observe
user id : 27
group abbr : doh
solvent abbr : DMSO
spectrometer id : 5
spectrometer name : Alec
spectrometer capacity : 60
```

```
Please select and option or enter -1 to quit:
2
Please enter the ID of the sample:
helloworld
Please enter a valid integer!
Please enter the ID of the sample:
1
Please enter the ID of the sample:
```

Remove sample:

Test Description	**Expected output**	**Actual output**	**Comment**
Search for valid sample	The program will remove the sample with that ID from the database	The program removed the sample with that ID from the database	Works as expected

Update test:

Test Description	**Expected output**	**Actual output**	**Comment**
Enter valid date	The data in the database will be successfully updates	Before  After 	Works as expected
Enter an invalid date	The program will ask the user to renter the date		Works as expected

Before

SAMPLE_ID	holder_no	record_submit_time
80001	3	2014-03-25 00:15:17

```
Please select and option or enter -1 to quit:
4
Please enter the ID of the sample:
80001
1. ExperimentName
2. ExperimentDescription
3. Holder no
4. Record submit time
5. Sample duration
6. Solvent
7. SpectrometerID
8. UserID
9. GroupName
Please choose a property you wish to update:
4
Please enter a valid time and date in the format YYYY-MM-DD HH:MM:SS
1996-09-03 09:00:00
Update done!
```

After

SAMPLE_ID	holder_no	record_submit_time
80001	3	1996-09-03 09:00:00

```
Please choose a property you wish to update:
4
Please enter a valid time and date in the format YYYY-MM-DD HH:MM:SS
1996/09/03 09:00:00
Invalid input
Please enter a valid time and date in the format YYYY-MM-DD HH:MM:SS
sfivf
Invalid input
Please enter a valid time and date in the format YYYY-MM-DD HH:MM:SS
1996/09/03 09:00:
Invalid input
Please enter a valid time and date in the format YYYY-MM-DD HH:MM:SS
```

Evaluation

After testing my program, I pleased to say it works as expected and is able to process large data files quickly and efficiently. I have used multiple classes and universal methods which take in various parameters making the program overall more modular and adaptable.

Conclusion

In conclusion I'm satisfied with end program, it is able to carry out the required tasks with ease and is able to handle erroneous data without crashing. The added extensions massively increase the overall functionality and makes the program far more flexible.