DSR Assignment 2 **z5215332**

Task 1 – Some Theory

a) Are the three tables in 1NF? Why or why not?

All the three tables are in 1NF because there is a primary key to uniquely ID data and there is one data item per cell.

b) Are the tables in 2NF? Why or why not?

All the three tables are in 2NF because all the non-key columns are dependent on the table's whole key.

c) Are the tables in 3NF? Why or why not?

Both the *Comments* and the *Replies* tables are not in 3NF because not all the non-key columns are dependent on the table's primary key. In both the *Comments* and the *Replies* table, the *storyDate*, is not dependent on the *commentID* (the primary key) as they are dependent on *storyID* (the foreign key) and are therefore redundant data.

The *Story* table is in 3NF because it contains only columns that are non-transitively dependent on the primary key.

Task 2 – System Design

DiggDBAccess
- Connection conn1 - Connection conn2
+ DiggDBAccess() + String printSimpleStatistics() - String printPartA() - String printPartB() - String printPartC() - String printPartD() - String printPartE() + void createUserTable() - void insertUploaderID

Originally, the printSimpleStatistics method would have to contain all the code for parts a – e, of Task 3 which is highly inefficient. Therefore, separate methods for each part were implemented in order to divide the longer method into smaller ones. Another benefit is that testing was made more efficient as any errors were thrown within a specific method so could be discovered and fixed much easier. By implementing each SQL as a separate method from each part of Task 3 (a-e), I was able to work through the SQL's individually and test throughout.

The same was done for the two parts of Task 4 – to allow for easier testing and more efficient code, with a second connection made to incorporate my own database.

Task 3 – Printing Simple Statistics

a) `SELECT DISTINCT category2 FROM Stories;`
b) `SELECT storyID, title, numDiggs, category2 FROM Stories WHERE numDiggs < 200;`
c) `SELECT AVG(numDiggs) FROM Stories;`
d) `SELECT DISTINCT uploaderID, COUNT(*) FROM Stories GROUP BY uploaderID HAVING COUNT(*) > 10;`
e) `SELECT commentID, numFollowers, Comments.storyID, originalURL FROM Comments LEFT JOIN Stories ON Comments.storyID = Stories.storyID WHERE numFollowers > 1000;`

Testing:

No errors were encountered when implementing the SQL (other than having to remember to open Putty to access the database).

However, originally Task 3 – Part C was misinterpreted by thinking that the average number of diggs was being displayed per story with the following SQL implemented;

SELECT storyID, AVG(numDiggs) FROM Stories GROUP BY storyID;

After talking to Saber, this interpretation was incorrect, and changes were made to the SQL that is now seen above.

Task 4 – Creating a User Table

a) Creating the table:

Fields (with datatypes) to be included in the table:

- uploaderID (varchar - 15) not null, unique \\ Primary Key
- first_name (varchar – 255) not null
- last_name (varchar – 255) not null
- age (int)
- gender (char – 1) not null
- dateCreated (date) not null – default of current date
- postcode (int)
- numPosts (int)
- email (varchar - 100) not null

Assumptions:

- all users will give both a first and last name, gender, creation date of user identity and an email address
- using ISO standards, the following four codes would be used for choosing gender: 0 = not known, 1 = male, 2 = female, 9 = not applicable
- The default of the current date will be filled for any user when creating their profile if another date has not already been entered

b) SQL Query to implement the table:

```
CREATE TABLE knownUsers (uploaderID varchar(15) UNIQUE, first_name VARCHAR(255), last_name varchar(255), age int, postcode int, numPosts int, gender CHAR(1), dateCreated timestamp DEFAULT CURRENT_TIMESTAMP NOT NULL, email varchar(100) UNIQUE, PRIMARY KEY (uploaderID));
```

Note: the above plan (in part a) could not be accurately implemented as the database is not being fully populated and therefore NOT NULL could not be used to create the table as it does not allow the INSERT INTO command to work in part c. Below is the create tablequery that would be used with NOT NULL in order to correctly implement the part a plan.

```
CREATE TABLE knownUsers (uploaderID varchar(15) NOT NULL UNIQUE, first_name VARCHAR(255) NOT NULL, last_name varchar(255) NOT NULL, age int, postcode int, numPosts int, gender CHAR(1) NOT NULL, dateCreated timestamp DEFAULT CURRENT_TIMESTAMP NOT NULL, email varchar(100) NOT NULL UNIQUE, PRIMARY KEY (uploaderID));
```

c) Populating the primary key (from stories, comments and replies)

```
INSERT INTO knownUsers(uploaderID) SELECT uploaderID AS uploaderID FROM dsr.Stories UNION SELECT Comments.userID AS uploaderID FROM dsr.Comments UNION SELECT Replies.userID AS uploaderID FROM dsr.Replies;
```

This SQL Query selects all uploaderID and userID fields from both Stories, Comments and Replies (39346 userID's) and inserts them all into the uploaderID column of the knownUsers table.

The following SQL Queries show the attempts made on Putty to add all the data separately before discovering and utilising UNION.

```
INSERT INTO knownUsers(uploaderID) SELECT DISTINCT userID FROM dsr.Comments; - 0.67 sec (30577 rows)
INSERT INTO knownUsers(uploaderID) SELECT  DISTINCT uploaderID FROM dsr.Stories; - 0.01 sec (391 rows)
INSERT INTO knownUsers(uploaderID) SELECT DISTINCT userID FROM dsr.Replies; - 0.51 sec (21399 rows)
```

Not only did it take significantly longer to create with the overall times of all three queries taking 1.19 seconds, but I also encountered issues when trying to add duplicates. The query that was finally decided on, only took 0.67 seconds to run and has no duplicates without the requirement for a distinct clause.

Task 5

Unit Testing - Overview of Tests Results:

The SQL Queries involving creating a table and inserting a certain set of data (as outlined in Task 4) were tested using putty and eclipse, altering the DiggDBAccess java class. The following tests were executed:

ID	Description	Input	Expected Output	Actual Output	Findings
1	The table has been added to the database	`System.out.println("Created table in given database...");`	Created table in given database…	`Created table in given database...`	Pass
2	The uploaderID's have been inserted into the database	`System.out.println("Inserted uploaderID's...");`	Inserted uploaderID's…	`Inserted uploaderID's...`	Pass

3	Retrieve an uploaderID from the database	`System.out.println(result);`	_skin_	`Task 4:`
				`Display contents:`
				`uploaderID = _skin_`
				`_skin_`
					Pass

4	Try and return a non-existent uploaderID	`System.out.println(result);`	Nothing printed cause SQL query returns nothing	`Task 4:`
				`Display contents:`
				`uploaderID not found`
					Pass

```
Used for testing purposes:
            ResultSet rs = stmt.executeQuery("SELECT uploaderID FROM knownUsers WHERE uploaderID = '_skin_'");
            String result += "Display contents: uploaderID = _skin_\n";
            while (rs.next())
            { result += (rs.getString("uploaderID") + "\n"); }
            System.out.println(result);
            rs = stmt.executeQuery("SELECT uploaderID FROM knownUsers WHERE uploaderID = '_skin1_'");
            result = "\n\nTask 4:\”; result += "Display contents: uploaderID not found\n";
            while (rs.next())
            { result += (rs.getString("uploaderID") + "\n"); }
            rs.close(); return result;
```

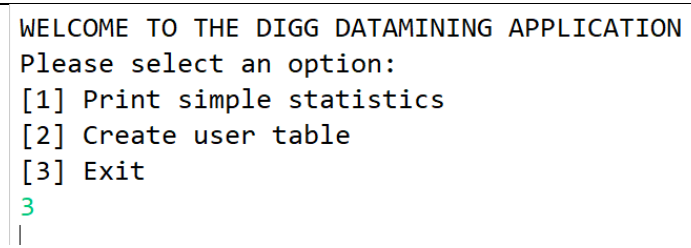
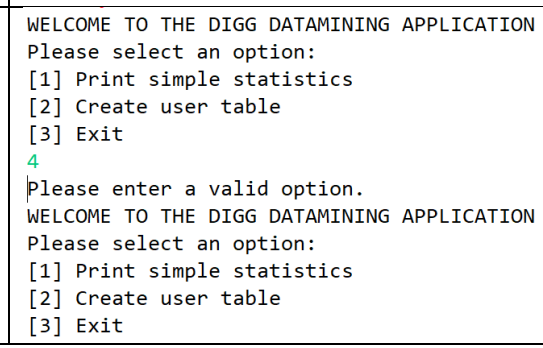
End User Testing –

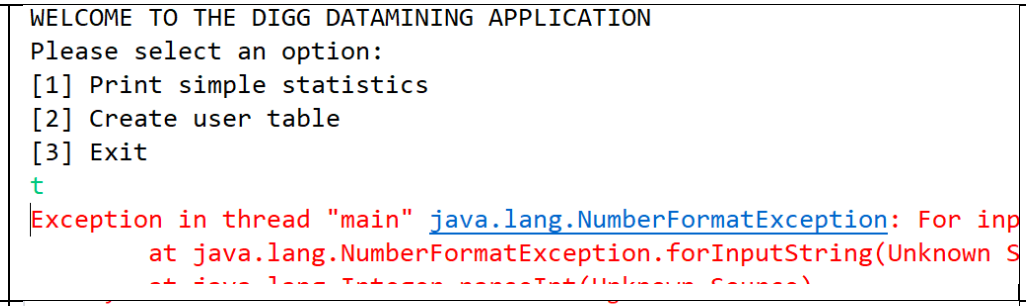
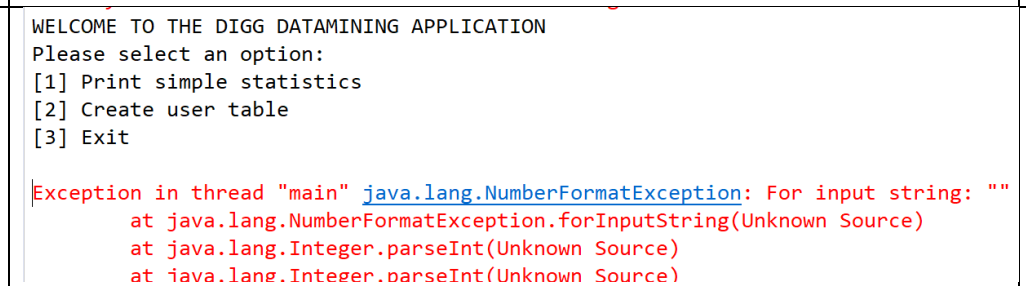
The end user testing will focus on the functions that the user will input to have the desired output.

```
WELCOME TO THE DIGG DATAMINING APPLICATION
Please select an option:
[1] Print simple statistics
[2] Create user table
[3] Exit
```

ID	Description	Input	Expected Output	Actual Output	Findings
1	User clicks ‘1’ from the option menu	‘1’	The simple statistics method from the DiggDBAccess class should run with all 5 queries printing to the screen.	Part A - Task 3: Display contents: category2 Technology Lifestyle Science Entertainment Gaming World & Business Offbeat Sports Part B - Task 3: Display contents: storyID, title, numDiggs, category2 9486052, New York in the 1930s (Photoset), 128, Lifestyle 9494200, Colombia tests OLPC laptops... running Windows XP, 183, Technology 9496223, Luhrmann's Australia 'unfinished', 86, Entertainment	Pass

				Part C -- Task 3: Display contents: average numDiggs 1289.6658 Part D -- Task 3: Display contents: uploaderID, number of stories AFelsinger 15 albyliontk 20 anderzole 14 badwithcomputer 41 bixby1 11 brainnovate 19 BrokebackCasket 12 Bukowsky 12 Burento 15 d2002 26 DirectTulip 21 gamebittk 24 Part E - Task 3: Display contents: commentID, numFollowers, storyID, originalURL 20295296 1002 9309213 http://www.chicagotribune.com/news/chi-halloween20081031225751,0,3914601.photo 20359461 1054 9341843 http://www.collegehumor.com/video:1888086 20370948 1004 9348676 http://www.fivethirtyeight.com/2008/11/todays-polls-113-pm-edition.html 20409758 1352 9367395 http://i35.tinypic.com/11iisz5.jpg 20413751 2884 9367395 http://i35.tinypic.com/11iisz5.jpg 20414116 2009 9367395 http://i35.tinypic.com/11iisz5.jpg 20415400 1292 9370604 http://i33.tinypic.com/zx6yi8.jpg 20459386 1614 9390056 http://farm4.static.flickr.com/3279/3006416363_95ef8de914_o.jpg 20503067 1006 9409228 http://forums2.battleon.com/f/tm.asp?m=15024647 20527922 1390 9426618 http://www.break.com/index/hot-college-chick-pulls-classic-prank.html 20588183 1341 9458873 http://www.collegehumor.com/video:1886349 20591180 1417 9458873 http://www.collegehumor.com/video:1886349 20592913 1300 9461582 http://www.theonion.com/content/node/43029 20607791 2012 9467640 http://www.vulomedia.com/images/25464truckfirezp5.jpg 20608800 1249 9467640 http://www.vulomedia.com/images/25464truckfirezp5.jpg	
2	User clicks ‘2’ from the option menu	‘2’	The createUserTable method should run (then the insert uploaderID method) with the user being notified	Please select an option: [1] Print simple statistics [2] Create user table [3] Exit 2 Created table in given database... Inserted uploaderID's...	Pass

			that a table has been created and the uploaderID's being inserted.		
3	User clicks '3' from the option menu	'3'	The program should exit.		Pass - an improvement to the DiggDBUserInterface design would be to include a statement that says the user has exited successfully (display 'Program ended.' etc.).
4	The user inputs an invalid number	'4'	If an incorrect value is entered by the user, they are prompted to ask again		Pass

5	The user enters a letter/symbol	‘t’	Error message thrown and another attempt allowed		Fail – a letter or symbol is not handled by an error statement and would have to be fixed when updating the User Interface
6	The user presses the enter key	Enter key is pressed	Exception is handled and another attempt is allowed		Fail – the enter key throws an exception for the user. This would have to be changed by a try and catch statement to let the user try again

Manipulation of the User Interface:

To further test the system, changes could be made to the DiggDBUserInterface class to allow the user to enter their own sets of data into the table and alter other sets.

By creating further switch statements under the create table option to then add, edit or delete elements of the table, the user would have greater access and operability with the database.

For example, if the user entered their own set of details to be added to the table (after altering the user interface – asking for the details from the user), the following SQL Query would run, and a new user could be added:

INSERT INTO knownUsers (uploaderID, first_name, last_name, age, postcode, numPosts, gender, dateCreated, email) VALUES (‘_bridget_free’, ‘Bridget’, ‘Free’, ‘18’, ‘2610’, ‘1’, ‘F’, ‘’, ‘bridget.e.free@gmail.com’);