SE3309a - Assignment Three

Samuel Mallabone - 250844429
Robert Northmore - 250838145
Jak Terpak - 250846924
Craig Cook - 250866685

Note: We also created a Github repository name SE3309Databases and included all the TA's in it. Github link: https://github.com/sam-mallabone/SE3309Databases (If this link doesn't work, the Github Repository's name is SE3309ADatabases and my username is sam-mallabone, here is a clonable link for the repository too: https://github.com/sam-mallabone/SE3309Databases.git)

Question 2: Creating Tables:

The top three screenshots show the commands to create the tables. All these commands ran properly and created the tables as we intended. Below the create table statements, we used the describe <table_name> command to show that the tables were successfully created and had the schema we intended.

```
CREATE TABLE book(
    ISBN BIGINT PRIMARY KEY,
    title VARCHAR(50) NOT NULL,
    publishingDate DATE,
    genre VARCHAR(20) NOT NULL,
    inventory INT NOT NULL,
    authorID INT NOT NULL,
    FOREIGN KEY(authorID) REFERENCES author(authorID)
);
```

```
CREATE TABLE user(
    ID INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30) NOT NULL,
    email VARCHAR(20) NOT NULL,
    age INT,
    phoneNumber VARCHAR(15),
    membershipStart DATE,
    membershipEnd DATE
);
```

```
CREATE TABLE employee(
    staffID INT AUTO_INCREMENT PRIMARY KEY,
    startDate DATE NOT NULL,
    salary INT NOT NULL,
    email VARCHAR(30) NOT NULL,
    phoneNumber VARCHAR(15) NOT NULL,
    name VARCHAR(30) NOT NULL,
    role VARCHAR(15) NOT NULL
);
```

```sql
CREATE TABLE userpaymentinfo (
    cardNumber BIGINT,
    userID INT NOT NULL,
    cardType VARCHAR(20) NOT NULL,
    expiryDate DATE NOT NULL,
    securityCode INT(3) NOT NULL,
    PRIMARY KEY (cardNumber),
    FOREIGN KEY (userID) REFERENCES user(ID)
);


CREATE TABLE returned (
    borrowID INT PRIMARY KEY,
    userID INT NOT NULL,
    bookID BIGINT NOT NULL,
    dateBorrowed DATE NOT NULL,
    expectedReturnDate DATE NOT NULL,
    dateReturned DATE NOT NULL,
    rating INT,
    FOREIGN KEY (userID) REFERENCES user(ID),
    FOREIGN KEY (bookID) REFERENCES book(ISBN)
);
```

```sql
CREATE TABLE outgoing
(
    borrowID              INT NOT NULL AUTO_INCREMENT,
    userID                INT NOT NULL,
    bookID                BIGINT NOT NULL,
    dateBorrowed          DATE NOT NULL,
    expectedReturnDate    DATE NOT NULL,
    PRIMARY KEY           (borrowID),
    FOREIGN KEY (userID) REFERENCES user(ID),
    FOREIGN KEY (bookID) REFERENCES book(ISBN)
);

CREATE TABLE author
(
    authorID              INT NOT NULL AUTO_INCREMENT,
    authorName            VARCHAR(30) NOT NULL,
    PRIMARY KEY           (authorID)
);

CREATE TABLE  payment  (
    borrowID INT NOT NULL,
    amountOwed DECIMAL(6,2),
    isPaid BOOLEAN,
    PRIMARY KEY (borrowID),
    FOREIGN KEY (borrowID) REFERENCES returned(borrowID)
);

CREATE TABLE wishlist (
    requestID INT NOT NULL,
    authorID INT,
    genre VARCHAR(20),
    title VARCHAR(50),
    userID INT NOT NULL,
    PRIMARY KEY (requestID),
    FOREIGN KEY (userID) REFERENCES user(ID),
    FOREIGN KEY (authorID) REFERENCES author(authorID)
);
```

This section is when we were running the describe <table_name> commands

Field	Type	Null	Key	Default	Extra
ISBN	bigint(20)	NO	PRI	NULL	
title	varchar(50)	NO		NULL	
publishingDate	date	YES		NULL	
genre	varchar(20)	NO		NULL	
inventory	int(11)	NO		NULL	
authorID	int(11)	NO	MUL	NULL	

Field	Type	Null	Key	Default	Extra
staffID	int(11)	NO	PRI	NULL	auto_increment
startDate	date	NO		NULL	
salary	int(11)	NO		NULL	
email	varchar(30)	NO		NULL	
phoneNumber	varchar(15)	NO		NULL	
name	varchar(30)	NO		NULL	
role	varchar(15)	NO		NULL	

Field	Type	Null	Key	Default	Extra
borrowID	int(11)	NO	PRI	NULL	auto_increment
userID	int(11)	NO	MUL	NULL	
bookID	bigint(20)	NO	MUL	NULL	
dateBorrowed	date	NO		NULL	
expectedReturnDate	date	NO		NULL	

Field	Type	Null	Key	Default	Extra
borrowID	int(11)	NO	PRI	NULL	
amountOwed	decimal(6,2)	YES		NULL	
isPaid	tinyint(1)	YES		NULL	

Field	Type	Null	Key	Default	Extra
borrowID	int(11)	NO	PRI	NULL	
userID	int(11)	NO	MUL	NULL	
bookID	bigint(20)	NO	MUL	NULL	
dateBorrowed	date	NO		NULL	
expectedReturnDate	date	NO		NULL	
dateReturned	date	NO		NULL	
rating	int(11)	YES		NULL	

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	auto_increment
name	varchar(30)	NO		NULL	
email	varchar(20)	NO		NULL	
age	int(11)	YES		NULL	
phoneNumber	varchar(15)	YES		NULL	
membershipStart	date	YES		NULL	
membershipEnd	date	YES		NULL	

Field	Type	Null	Key	Default	Extra
cardNumber	bigint(20)	NO	PRI	NULL	
userID	int(11)	NO	MUL	NULL	
cardType	varchar(20)	NO		NULL	
expiryDate	date	NO		NULL	
securityCode	int(3)	NO		NULL	

Field	Type	Null	Key	Default	Extra
requestID	int(11)	NO	PRI	NULL	
authorID	int(11)	YES	MUL	NULL	
genre	varchar(20)	YES		NULL	
title	varchar(50)	YES		NULL	
userID	int(11)	NO	MUL	NULL	

Field	Type	Null	Key	Default	Extra
authorID	int(11)	NO	PRI	NULL	auto_increment
authorName	varchar(30)	NO		NULL	

Question 3: Three different types of Insert

*Below shows the three different insert commands for entering data into tables. All three of these different types of commands worked for us and the screenshots show both the command and the output. Instead of running the select * statement for the table to show that the values had been entered, we showed the success statement of the output because we thought this would exemplify that the command worked as intended.*

```
INSERT INTO user VALUES (
    0,
    'Robert Northmore',
    'rnorthm@uwo.ca',
    20,
    "705-464-7288",
    DATE '2017-10-14',
    DATE '2020-10-14'
);
```

32	11:43:51	select * from user LIMIT 0, 1000		1 row(s) returned
33	11:44:38	INSERT INTO user VALUES (0, 'Sam Mallabone', 'smallabo@uwo.ca', 20, "403-992-7946", DAT...		1 row(s) affected
34	11:45:05	INSERT INTO user VALUES (0, 'Jakson Terpak', 'jterpak@uwo.ca', 20, "905-395-6816", DATE '...		1 row(s) affected
35	11:45:40	INSERT INTO user VALUES (0, 'Robert Northmore', 'rnorthm@uwo.ca', 20, "705-464-7288", DA...		1 row(s) affected

```
INSERT INTO book (
    ISBN,
    title,
    publishingDate,
    genre,
    inventory,
    authorID
) VALUES (
    9780545010221,
    'Harry Potter and the Deathly Hallows',
    DATE '2007-07-21',
    'Fantasy Fiction',
    25,
    1
);
```

✓	40	11:54:17	INSERT INTO book (ISBN, title, publishingDate, genre, inventory, authorID) VALUES (9780545...	1 row(s) affected
✓	41	11:54:36	select * from book LIMIT 0, 1000	1 row(s) returned

```
-- the command commented out was to enter an entry into the database that could then be used in the insert
INSERT INTO outgoing
VALUES
(1106, 1683, 3024002041378, DATE '2017-09-18', DATE '2017-12-23');

INSERT INTO returned (
        borrowID,
        userID,
        bookID,
        dateBorrowed,
        expectedReturnDate,
        dateReturned,
        rating
    ) SELECT borrowID, userID, bookID, dateBorrowed, expectedReturnDate, DATE '2017-11-22', 4 FROM outgoing
WHERE expectedReturnDate=DATE '2017-12-23';
```

Output

Action Output

#	Time	Action	Message
✓ 9	11:37:51	select * from outgoing LIMIT 0, 10000	104 row(s) returned
✓ 10	11:47:19	INSERT INTO outgoing VALUES (1106, 1683, 3024002041378, DATE '2017-09-18', DATE '2017-12-23')	1 row(s) affected
✓ 11	11:47:24	INSERT INTO returned (borrowID, userID, bookID, dateBorrowed, expectedReturnDate, dateReturned, ra...	1 row(s) affected Records: 1 Duplicates: 0 Warnings: 0

Question 4: Inserting data into the tuples.

To insert our data into the tuples, we used python (the files name is insertmysql.py). This python file would run the script that enters data into the database. We ensured to create the tables in such a way that we could use existing primary keys from a created table for the foreign keys on another table. This ensured that we could properly use all the joins and that our queries would run as expected. In the python file a lot of the code is commented out and in weird order. We ran several sections of the code to insert data which accounts for the seemingly random ordering of the code. It worked perfectly and we were able to enter over 5000 tuples into two tables, several hundred into another table and over one hundred tuples the remaining tables. To get our data, we used a combination of csv files containing names as well as a python library called Faker which allowed us to generate random names and random text which we took advantage of.

Question 5: Six Interesting queries

Below are screenshots of our six interesting queries and their outputs/results.

```
1  -- First Interest query getting all employees who have a salary over 30000 or have role security
2  SELECT name, email, role
3  FROM employee WHERE salary > 30000 OR role = 'security'
4  ORDER BY role;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	email	role
Jeffery Watson	Aaron@u.ca	assistant
Kathleen Giles	Abel@u.ca	assistant
Michael Hamilton	Gus@u.ca	assistant
Matthew Cooper	Gwen@u.ca	assistant
Teresa Turner	Gwendolyn@u.ca	assistant
Donald Stafford	Gudrun@u.ca	assistant
Megan Henson	Guy@u.ca	assistant
Matthew Bruce	Guy@u.ca	janitor
Paul Mays	Guillermo@u.ca	janitor
Miss Victoria Anderson	Gussie@u.ca	janitor
Christopher Mitchell	Gwenda@u.ca	janitor
Luis Molina	Gwendolyn@u.ca	janitor
Nicole Mendez	Aaron@u.ca	librarian
Joseph Wilcox	Abby@u.ca	librarian
Julie Bishop	Abdul@u.ca	librarian
Nathan Reeves	Abigail@u.ca	librarian
Kathleen Rodriguez	Gudrun@u.ca	librarian
Gregory Alexander	Gussie@u.ca	librarian
Amanda Perez	Gustavo@u.ca	librarian
Christine Mccarthy	Guillermina@u.ca	librarian
Elizabeth Smith	Gustavo@u.ca	librarian
Tyler Sanchez	Aaron@u.ca	security
John White	Abbey@u.ca	security
Willie Mercer	Abbie@u.ca	security
Tara Garcia	Guillermina@u.ca	security

```
6  -- get the users with outgoing romance books and that are over age 35
7  SELECT user.name, user.email, user.age, book.title
8  FROM outgoing
9  INNER JOIN book ON book.ISBN = outgoing.bookID
10 INNER JOIN user ON user.ID = outgoing.userID
11 WHERE user.age > 35 AND book.genre = 'romance'
12 ORDER BY user.name;
13
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	email	age	title
Bartuska Eloise	Bartuska@ola.ca	73	Expedita moll
Blumenstein Ja	Blumenstein@ola.ca	73	Deleniti ulla
Gaieski	Gaieski@ov.com	39	Laboriosam re
Goshorn	Goshorn@nik.ca	45	Accusantium a
Hilgeman	Hilgeman@ola.ca	50	Dolorum natus
Keplev	Keplev@ola.ca	60	Consequuntur

Note: a count was returned

```
14  -- Get amount of users under the age of 35 with overdue books where they owe more than one dollar and haven't paid yet
15  SELECT COUNT(*) as UsersWithOverdueBooks
16  FROM returned
17  INNER JOIN payment ON returned.borrowID = payment.borrowID
18  INNER JOIN user ON returned.userID = user.ID
19  WHERE user.age < 35 AND payment.amountOwed > 1.00 AND payment.isPaid = 0;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

UsersWithOverdueBooks
5

```
21  -- Get the book title, author name, genre and inventory of the books with under 10 in stock, and that are of type adventure or horror.
22  -- returned in decreasing order by inventory
23  SELECT book.title, author.authorName, genre, book.inventory
24  FROM book
25  NATURAL JOIN author
26  WHERE book.inventory < 10 AND (book.genre = 'horror' OR book.genre='adventure')
27  ORDER BY book.inventory DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

title	authorName	genre	inventory
Eaque autem o	Melissa Whitney	adventure	9
Corporis dolo	Russell Brown	horror	9
Corporis quia	Catherine Bover	horror	9
Iosam fugiat	Ashley Barron	adventure	8
Omnis totam a	Ashley Marsh	adventure	8
Reprehenderit	David Rodriguez	horror	8
Sequi esse au	Brian Perez	adventure	8
Rerum commodi	Sean Mcdonald	adventure	8
Sequi ab odit	Kellie Brown	adventure	8
Animi culpa o	Linda Bell	adventure	6
Nobis quia no	David Clarke	adventure	6
Quo natus ost	Gina Brooks	adventure	6
Magnam fuga o	Jasmine Woods	adventure	6

```
29  -- get authors who's names start with an and the books they have written that are present within the library.
30  -- order by authors with a book present. Left join so authors without a book are still included in the query.
31  SELECT author.authorID, author.authorName, book.title, book.genre
32  FROM author
33  LEFT JOIN book ON author.authorID = book.authorID
34  WHERE author.authorName LIKE 'an%' ORDER BY book.title DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

authorID	authorName	title	genre
2022	Angelica Giles	Voluptates ma	horror
4993	Andrew Wilson	Voluptates il	fiction
649	Anna Mcmahon	Rerum volupta	fiction
4924	Anthony Osborne	Ratione tempo	mystery
1304	Andrew Taylor	Quidem volupt	mystery
3694	Anthony Schwartz	Quia ad porro	comedy
2442	Anna Lester	Qui magni dol	satire
2766	Angela Wilson	Provident sol	scientific
3695	Anthony Cardenas DVM	Placeat esse	adventure
1432	Angela Smith	Perspiciatis	non-fiction
2247	Angela Drake	Nesciunt odit	non-fiction
1278	Anthony Phillips	Necessitatibu	satire
3171	Andre Butler	Itaque quae c	adventure

Note: Only the one tuple was returned

```
36  -- Get the person's details who owes the maximum amount of money to the library
37  SELECT user.name, user.email, user.phoneNumber, MAX(payment.amountOwed) as OwedToLibrary
38  FROM returned
39  INNER JOIN payment ON returned.borrowID = payment.borrowID
40  INNER JOIN user ON returned.userID = user.ID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name	email	phoneNumber	OwedToLibrary
Bawner Georgianne	Bawner@ola.ca	992-260-8404	8.50

Question 6: Delete and Update

Below shows three interesting modification commands. We choose to do one interesting delete and two interesting updates. For each of these commands we included the MySQL statement and the output from running the command to show that the command worked as intended.

Delete:

```
2  --  Delete from employees all security or people who's name start with Mat
3  DELETE FROM employee WHERE role = 'security' OR name LIKE 'Mat%';
```

```
30  12:20:10  DELETE FROM employee WHERE role = 'security' OR name LIKE 'Mat%'  5 row(s) affected
```

Update:

```
5  -- Extending the amount of time the book is borrowed for all books that have to be returned on a certain day
6  UPDATE outgoing SET expectedReturnDate = DATE '2018-01-02' WHERE expectedReturnDate = DATE '2017-12-29';
```

```
33  12:27:01  UPDATE outgoing SET expectedReturnDate = DATE '2017-12-29' WHERE expectedReturnDate = DATE '20...  4 row(s) affected Rows matched: 4  Changed: 4  Warnings: 0
```

Update 2:

```
18  -- Update the dateborrowed of a book who's date borrowed was 2017-10-22 based on the query results of the returned table whos
19  -- expected return date is 2017-11-28
20  UPDATE outgoing as dest, (SELECT * FROM returned where expectedReturnDate = DATE '2017-11-28') as src
21  SET dest.dateBorrowed = src.dateBorrowed WHERE dest.dateBorrowed = DATE '2017-10-22';
```

```
39  12:43:43  UPDATE outgoing as dest, (SELECT * FROM returned where expectedReturnDate = DATE '2017-11-28') as sr...  1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0
```

Question 7: Views

Below shows the View commands and the result tables from running the commands. We ran a Create View command and then a select * command was run to see all the tuples within the View.

```sql
-- Creates a view of all employees who are librarians
CREATE VIEW Librarians AS
SELECT * FROM Employee
WHERE role = 'librarian'
ORDER BY name;
```

Result from querying all tuples in the above view

staffID	startDate	salary	email	phoneNumber	name	role
33	2013-01-19	41000	Gustavo@u.ca	334-119-1778	Amanda Perez	librarian
39	2013-09-09	41000	Guillermina@u.ca	334-129-1826	Christine Mccarthy	librarian
43	2013-09-09	41000	Gustavo@u.ca	334-129-1878	Elizabeth Smith	librarian
32	2013-01-19	41000	Gussie@u.ca	334-119-1765	Gregory Alexander	librarian
14	2017-05-19	24000	Abdul@u.ca	301-159-1091	Janice Gonzalez	librarian
23	2016-01-19	35000	Abby@u.ca	301-159-1078	Joseph Wilcox	librarian
24	2016-01-19	35000	Abdul@u.ca	301-159-1091	Julie Bishop	librarian
28	2013-01-19	41000	Gudrun@u.ca	334-119-1713	Kathleen Rodriguez	librarian
11	2017-05-19	24000	Abbey@u.ca	301-159-1052	Margaret Beltran	librarian
51	2017-12-19	42500	guy4@me.com	401-991-9243	Matty Johnson	librarian
27	2016-01-19	35000	Abigail@u.ca	301-159-1130	Nathan Reeves	librarian
20	2016-01-19	35000	Aaron@u.ca	301-159-1039	Nicole Mendez	librarian

VIEW 2:

```sql
-- Second view created
-- Create a view of all the books in the library who have been published in 2017
CREATE VIEW NewBooks(bookName, authorName, genre, publishingDate) AS
SELECT book.title, author.authorName, book.genre, book.publishingDate
FROM book
NATURAL JOIN author
WHERE book.publishingDate > DATE '2017-01-01'
ORDER BY book.genre;
```

Result from querying all tuples within the view:

bookName	authorName	genre	publishingDate
Quisquam ad d	Cassandra Nash	comedy	2017-05-13
Velit voluota	Allen Andrews	comedy	2017-05-13
Expedita anim	Ashley Galvan	comedy	2017-05-13
Ratione sit e	Amanda Jenkins	comedy	2017-05-13
Rem qui autem	Adam Johnson	comedy	2017-05-13
Quam consequa	Lauren Larson	comedy	2017-05-13
Quia animi si	David Lyons	fiction	2017-05-13
Voluptates il	Andrew Wilson	fiction	2017-05-13
Sunt id porro	Kimberly Cisneros	fiction	2017-05-13
Esse laborum	Jasmin Small	fiction	2017-05-13

VIEW 3:

```
21  -- View of all authors who have their books currently in the library
22  -- IE this excludes all authors who are in the library database but don't have their book in the library currently
23  CREATE VIEW AuthorsInLibrary (name, ID) AS
24  SELECT author.authorName, author.authorID
25  FROM author
26  INNER JOIN book ON author.authorID = book.authorID
27  ORDER BY author.authorName;
```

Resulting from querying all rows within the view.

name	ID
Adam Davis	454
Adam Johnson	3040
Adam Morales	3275
Adam Ortiz	3991
Adam Patterson	2918
Adam Robles	4821
Adam Smith	550
Adam Smith	550
Adrian Williams	2494
Adriana Thomas	1051

Inserting into the views we created:

Note: see the comments above the command for an explanation of why the insertion worked or did not work.

Working insert for view 1:

```
31    -- this works because this view is composed of only one table
32 •  INSERT INTO Librarians VALUES (0, DATE '2015-02-12', 35360, 'g@owl.ca', '778-998-2939', 'Jack Johanis', 'librarian');
```

```
49  13:03:55  INSERT INTO Librarians VALUES (0, DATE '2015-02-12', 35360, 'g@owl.ca', '778-998-2939', 'Jack Johanis', 1...  1 row(s) affected
```

Not working insert for view 2:

```
34    -- this won't work because this view is composed of multiple tables
35 •  INSERT INTO NewBooks VALUES ('Jack and the Magic Hat', 'Jackson Holt', 'adventure', DATE '2017-02-17');
```

```
50  13:08:37  INSERT INTO NewBooks VALUES ('Jack and the Magic Hat', 'Jackson Holt', 'adventure', DATE '2017-02-17')  Error Code: 1394. Can not insert into join view 'librarydatabase.newbooks' without fields list
```

The third view also will not be able to have values inserted because it is composed of a join between multiple tables. We chose not to include the screenshots of the failed insert because it is the same as the one above.

Question 8: Non-Existent MySQL clause.

The SQL clause that we learned in class, but was not implemented in MySQL is the assertion statement. An assertion is a predicate expressing a condition we wish the database to always satisfy. Through our experience with MySQL we found out that this assertion was unavailable and thus we would not be able to use it if necessary.