TIS1101 Database Fundamentals

Assignment 2

Title: University Library System

Prepared By:

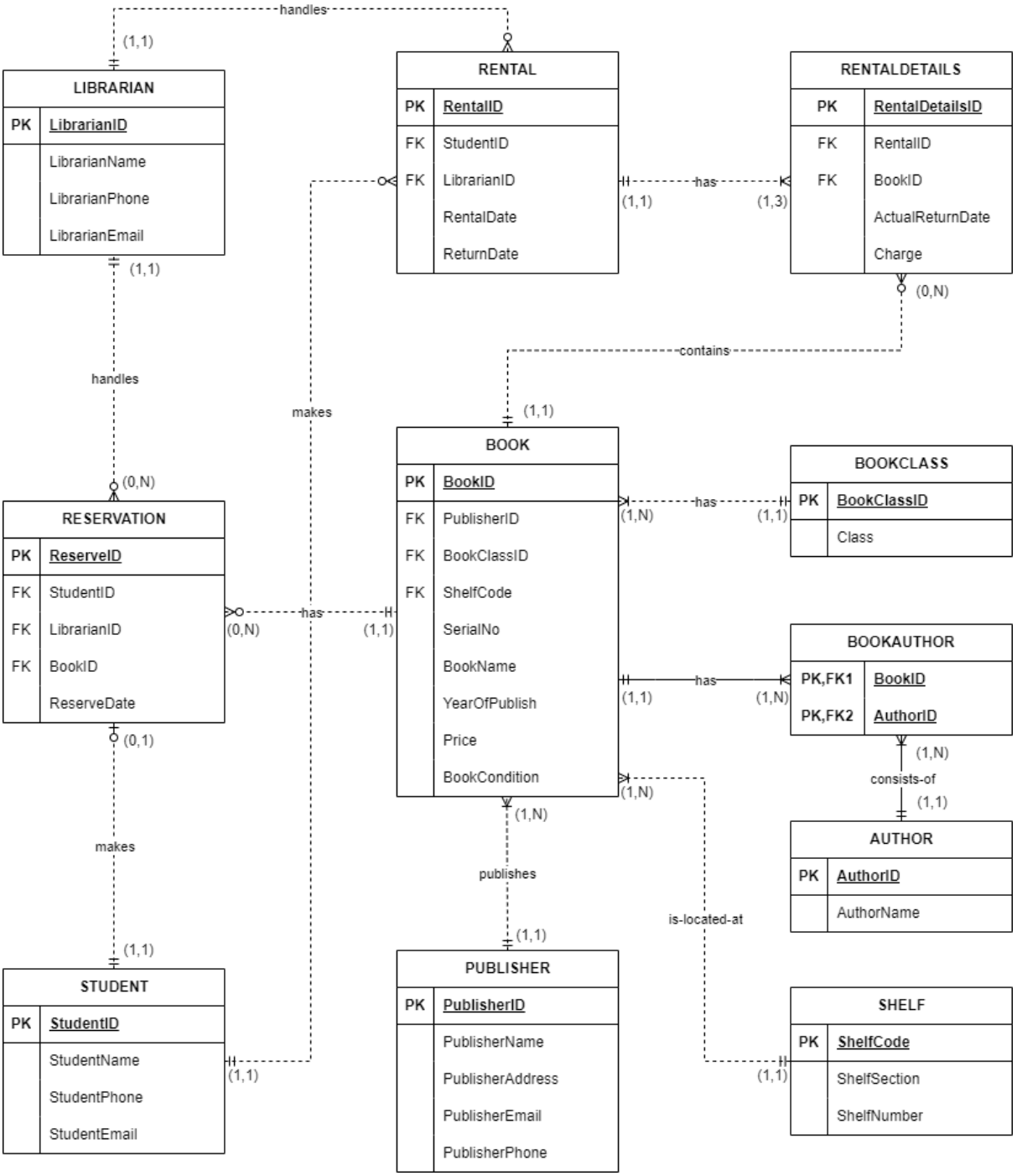
Leader:	1211103146	Cheryl Gwee En Xin	1211103146@student.mmu.edu.my
Member:	1221303203	Toh Ee Lin	1221303203@student.mmu.edu.my
	1221303636	Nabila Nadia binti Md Zaid	1221303636@student.mmu.edu.my
	1221303972	Lee Jia Ying	1221303972@student.mmu.edu.my

Table of Contents

Business Rules

1. A librarian may handle many books
2. Each rental has at most 3 books rented
3. Each loan transaction must have a separate record of the book loaned, return day of the book and the fee charged (if any)
4. The loan period for students are 21 days
5. Students are charged a RM2 fee for late return per day
6. Students are allowed to reserve only one book that is currently being loaned
7. A book may have many reservations
8. A book must be authored by at least one author; An author can author many books
9. Each book is categorised under a single book type (Science, Literature, Language, etc.)
10. Each book is published by a publisher; A publisher publishes many books
11. Each book belongs on a single shelf level on a shelving unit, which is located within a shelf section

Entity Relationship Diagram



Data Dictionary

TABLE NAME	ATTRIBUTE NAME	CONTENTS	TYPE	REQUIRED	PK OR FK	FK REFERENCED TABLE
LIBRARIAN	LibrarianID	Librarian unique identifier	Int	Y	PK	
	LibrarianName	Librarian name	Varchar(80)	Y		
	LibrarianPhone	Librarian phone number	Bigint	Y		
	LibrarianEmail	Librarian email	Varchar(50)	Y		
STUDENT	StudentID	Student unique identifier	Bigint	Y	PK	
	StudentName	Student name	Varchar(80)	Y		
	StudentPhone	Student phone number	Bigint	Y		
	StudentEmail	Student email	Varchar(50)	Y		
PUBLISHER	PublisherID	Publisher unique identifier	Int	Y	PK	
	PublisherName	Publisher name	Varchar(80)	Y		
	PublisherAddress	Publisher address	Varchar(150)			
	PublisherEmail	Publisher email	Varchar(50)	Y		
	PublisherPhone	Publisher phone number	Bigint	Y		

BOOKCLASS	BookClassID	Book class unique identifier	Int	Y	PK	
	Class	Book class	Varchar(50)	Y		
AUTHOR	AuthorID	Author unique identifier	int	Y	PK	
	AuthorName	Author name	Varchar(80)	Y		
SHELF	ShelfCode	Shelf code	Char(5)	Y	PK	
	ShelfSection	Shelf section	Int	Y		
	ShelfNumber	Shelf number	int	Y		
BOOK	BookID	Book unique identifier	Int	Y	PK	
	PublisherID	Publisher unique identifier	Int	Y	FK	PUBLISHER
	BookClassID	Book class unique identifier	Int	Y	FK	BOOKCLASS
	ShelfCode	Shelf code	Char(5)	Y	FK	SHELF
	SerialNo	Serial number of book	Bigint	Y		
	BookName	Book name	Varchar(80)	Y		
	YearOfPublish	Year of publish of book	Int			
	Price	Book price	Decimal(6,2)	Y		
	BookCondition	Book condition	Varchar(15)	Y		
BOOKAUTHOR	BookID	Book unique identifier	Int	Y	PK,FK1	BOOK
	AuthorID	Author unique identifier	Int	Y	PK,FK2	AUTHOR

RENTAL	RentalID	Book rental unique identifier	Int	Y	PK	
	StudentID	Student unique identifier	Int	Y	FK	STUDENT
	LibrarianID	Librarian unique identifier	Int	Y	FK	LIBRARIAN
	RentalDate	Book rental date	Date	Y		
	ReturnDate	Book return date	Date	Y		
RENTALDETAILS	RentalDetailsID	Book rental details unique identifier	Int	Y	PK	
	RentalID	Book rental unique identifier	Int	Y	FK	RENTAL
	BookID	Book unique identifier	Int	Y	FK	BOOK
	ActualReturnDate	Book actual return date	Date			
	Charge	Book late return charge	Decimal(6,2)			
RESERVATION	ReserveID	Book reserve unique identifier	Int	Y	PK	
	StudentID	Student unique identifier	Int	Y	FK	STUDENT
	LibrarianID	Librarian unique identifier	Int	Y	FK	LIBRARIAN
	BookID	Book unique identifier	Int	Y	FK	BOOK
	ReserveDate	Book reserve date	Date	Y		

Creation of Tables

Librarian

```
CREATE TABLE Librarian
(
        LibrarianID INT PRIMARY KEY NOT NULL
                GENERATED ALWAYS AS IDENTITY
                (START WITH 1, INCREMENT BY 1, NO CYCLE),
        LibrarianName VARCHAR(80),
        LibrarianPhone BIGINT,
        LibrarianEmail VARCHAR(50)
);
```

Table named 'Librarian' with LibrarianID as a unique primary key. Starting from a value of 1, the LibrarianID will increment by a value of 1 each time an entry to the librarian table is inserted instead of having to key in a manual LibrarianID. The value for LibrarianID must always be generated this way and cannot be overridden unless altered. The values required for this table are the names and contact information of the librarians, which consists of the LibrarianName, LibrarianPhone, and LibrarianEmail.

Student

```
CREATE TABLE Student
(
        StudentID INT PRIMARY KEY NOT NULL
                CONSTRAINT CK_Student_StudentID
                CHECK (StudentID > 1000000000 AND StudentID <2000000000),
        StudentName VARCHAR(80),
        StudentPhone BIGINT,
        StudentEmail VARCHAR(50)
);
```

Creates table Student which uses the natural key of StudentID assigned by the university. Dissimilar to the Librarian table above, the StudentID has to be entered manually. The StudentID will be confined between the values of 1 000 000 000 and 2 000 000 000 which are both numbers composed of 10 digits to help reduce the number of wrongly entered data. StudentName, StudentPhone and StudentEmail are required data in order to aid in the process of contacting the student.

Publisher

```
CREATE TABLE Publisher
(
        PublisherID INT PRIMARY KEY NOT NULL
                GENERATED ALWAYS AS IDENTITY
                (START WITH 1, INCREMENT BY 1, NO CYCLE),
        PublisherName VARCHAR(80),
        PublisherAddress VARCHAR(150),
        PublisherEmail VARCHAR(50),
        PublisherPhone BIGINT
);
```

Table containing all instances of the publishers that have books in the library. The primary key for the publisher is always generated automatically starting from the value of 1 and increments by 1 for convenience. Data included in this table are the publisher's name, address, email and phone number to contact them easily for any purchase of books or for other general purposes.

BookClass

```
CREATE TABLE BookClass
(
        BookClassID INT PRIMARY KEY NOT NULL
                CONSTRAINT CK_BookClass_Class
                CHECK ( BookClassID = 000 OR
                                BookClassID = 100 OR
                                BookClassID = 200 OR
                                BookClassID = 300 OR
                                BookClassID = 400 OR
                                BookClassID = 500 OR
                                BookClassID = 600 OR
                                BookClassID = 700 OR
                                BookClassID = 800 OR
                                BookClassID = 900
                                ),
        Class VARCHAR(50)
);
```

The BookClass table is a table for categorizing books into different genres. The categories here are separated using one of the world's most widely used library classification systems—the Dewey Decimal System. It consists of 10 different groups, with each group consisting of 100 numbers. But for the ease of creating the database, only the 10 main groups have been used. Using a constraint, it is ensured that the BookClassID primary key is only composed of those 10 main groups, with the Class field denoting the name of the group.

Author

```
CREATE TABLE Author
(
        AuthorID INT PRIMARY KEY NOT NULL
                GENERATED ALWAYS AS IDENTITY
                (START WITH 1, INCREMENT BY 1, NO CYCLE),
        AuthorName VARCHAR(80)
);
```

Author table contains all distinct individual authors that have written / co-authored a book in the library. A surrogate key is used here. The only other data field other than the primary key being used here is AuthorName which consists of the author's full name.

Shelf

```
CREATE TABLE Shelf
(
        ShelfCode CHAR(5) PRIMARY KEY NOT NULL,
        ShelfSection INT
                CONSTRAINT CK_Shelf_ShelfSection
                CHECK ( ShelfSection > 0 AND ShelfSection < 100 ),
        ShelfNumber INT
                CONSTRAINT CK_Shelf_ShelfNumber
                CHECK ( ShelfNumber > 0 AND ShelfNumber < 100 )
);
```

Unlike the other primary keys used in other tables, ShelfCode uses a data type CHAR. ShelfSection refers to different physical sections located in the library with many bookshelves. ShelfNumber refers to the different levels of each bookshelf. For this table, an upper limit of 100 has been implemented for both ShelfSection and ShelfNumber

Book

```
CREATE TABLE Book
(
        BookID INT PRIMARY KEY NOT NULL
                GENERATED ALWAYS AS IDENTITY
                (START WITH 1, INCREMENT BY 1, NO CYCLE),
        PublisherID INT,
        BookClassID INT,
        ShelfCode CHAR(5),
        SerialNo BIGINT,
        BookName VARCHAR(80),
        YearOfPublish INT
```

```
                CONSTRAINT CK_Book_YearOfPublish
                CHECK (YearOfPublish > 1000 OR YearOfPublish < 2500),
        Price DECIMAL(6,2),
        BookCondition VARCHAR(15) DEFAULT 'AVAILABLE'
                CONSTRAINT CK_Book_BookCondition
                CHECK (
                        BookCondition = 'AVAILABLE' OR --AVAILABLE FOR RENTAL--
                        BookCondition = 'NOT AVAILABLE' OR --NOT AVAILABLE FOR RENTAL--
                        BookCondition = 'LIB USE ONLY' OR --ONLY FOR LIBRARY USE--
                        BookCondition = 'RESERVED' OR --RESERVATIONID EXISTS FOR BOOKID--
                        BookCondition = 'MISSING' OR --BOOK IS REPORTED AS MISSING--
                        BookCondition = 'DAMAGED' OR --BOOK IS DAMAGED--
                        BookCondition = 'BILLED' --CHARGE IS GENERATED FOR BOOKID--
                        ),
        FOREIGN KEY (PublisherID) REFERENCES Publisher
                ON UPDATE NO ACTION
                ON DELETE NO ACTION,
        FOREIGN KEY (BookClassID) REFERENCES BookClass
                ON UPDATE NO ACTION
                ON DELETE NO ACTION,
        FOREIGN KEY (ShelfCode) REFERENCES Shelf
                ON UPDATE NO ACTION
                ON DELETE NO ACTION
);
```

BookID is made primary key here instead of SerialNo (the ISBN of the book which is unique for all books). This is because while ISBN code can help distinguish between books of similar titles, multiple copies of the same book may exist and an appropriate system to store information of the book copies does not exist. Thus a surrogate key is used here. Within the Book table, foreign key of PublisherID, BookClassID and ShelfCode is stored. Other information includes SerialNo, BookName, YearOfPublish and BookCondition—of which only seven conditions are allowed to be entered as a value.

BookAuthor

```
CREATE TABLE BookAuthor
(
        BookID INT NOT NULL,
        AuthorID INT NOT NULL,
        FOREIGN KEY (BookID) REFERENCES BOOK
                ON UPDATE NO ACTION
                ON DELETE NO ACTION,
        FOREIGN KEY (AuthorID) REFERENCES AUTHOR
                ON UPDATE NO ACTION
                ON DELETE NO ACTION,
        PRIMARY KEY (BookID, AuthorID)
);
```

The BookAuthor table bridges the gap between Author and Book. Multiple BookAuthor instances may exist for a single book and an author may exist within multiple BookAuthor instances. This allows a book to have more than one author and one author to write more than one book. BookID and AuthorID are used as composite keys for this table.

Rental

```
CREATE TABLE Rental
(
        RentalID INT PRIMARY KEY NOT NULL
                GENERATED ALWAYS AS IDENTITY
                (START WITH 1, INCREMENT BY 1, NO CYCLE),
        StudentID INT,
        LibrarianID INT,
        RentalDate DATE,
        ReturnDate DATE,
        FOREIGN KEY (StudentID) REFERENCES Student
                ON UPDATE NO ACTION
                ON DELETE NO ACTION,
        FOREIGN KEY (LibrarianID) REFERENCES Librarian
                ON UPDATE NO ACTION
                ON DELETE NO ACTION
);
```

Rental table records the details of each rental transaction. RentalID as a primary key is generated automatically for every insert, and it auto-increments by 1 each time. StudentID contains the data for the student who has proceeded with a book rental transaction. LibrarianID is for the librarian who handled the transaction. Other than that, RentalDate has been set to be input manually while the ReturnDate will be calculated automatically using a trigger that will be added later (pg. 35).

RentalDetails

```
CREATE TABLE RentalDetails
(
        RentalDetailsID INT PRIMARY KEY NOT NULL
                GENERATED ALWAYS AS IDENTITY
                (START WITH 1, INCREMENT BY 1, NO CYCLE),
        RentalID INT,
        BookID INT,
        ActualReturnDate DATE DEFAULT NULL,
        Charge DECIMAL(6,2) DEFAULT NULL,
        FOREIGN KEY (RentalID) REFERENCES Rental
                ON UPDATE NO ACTION
                ON DELETE NO ACTION,
        FOREIGN KEY (BookID) REFERENCES Book
                ON UPDATE NO ACTION
                ON DELETE NO ACTION
);
```

For each Rental instance, there exists a RentalDetail which contains the detailed information of each rental transaction. A maximum of 3 RentalDetail instances may exist for each Rental instance as only a maximum of 3 books can be borrowed by a student from the library. BookID are the books that have been borrowed, while ActualReturnDate denotes the date that the student returns the book. Charge will be a field that is automatically calculated if the student doesn't manage to return the book on time.

Reservation

```
CREATE TABLE Reservation
(
        ReserveID INT PRIMARY KEY NOT NULL
                GENERATED ALWAYS AS IDENTITY
                (START WITH 1, INCREMENT BY 1, NO CYCLE),
        StudentID INT,
        LibrarianID INT,
        BookID INT,
        ReserveDate DATE,
        FOREIGN KEY (StudentID) REFERENCES Student
                ON UPDATE NO ACTION
                ON DELETE NO ACTION,
        FOREIGN KEY (LibrarianID) REFERENCES Librarian
                ON UPDATE NO ACTION
                ON DELETE NO ACTION
);
```

Books can be reserved if another student is currently renting it. Thus the need for a Reserve table. ReserveID here is also generated for each insert into reservation by an increment of 1. StudentID, BookID and ReserveDate are used as fields here. Unlike the rental system, a student can only reserve a maximum of one book per time.

Insertion of Records

Provide SQL commands and screenshots of each table showing the inserted records
Maybe can do deletion of data

Librarian

Brief:

```
INSERT INTO Librarian
VALUES (
          DEFAULT,
          'Eleanora',
          60321481445,
          'eleanora@nnu.edu.my'
        );
```

Data Insertion:

```
INSERT INTO Librarian
VALUES (DEFAULT, 'Eleanora', 60321481445, 'eleanora@nnu.edu.my');
INSERT INTO Librarian
VALUES (DEFAULT, 'Judy', 60379563868, 'judy@nnu.edu.my');
INSERT INTO Librarian
VALUES (DEFAULT, 'Felicia', 60321480666, 'felicia@nnu.edu.my');
INSERT INTO Librarian
VALUES (DEFAULT, 'Brielle', 60356368656, 'brielle@nnu.edu.my');
INSERT INTO Librarian
VALUES (DEFAULT, 'Merton', 60362529712, 'merton@nnu.edu.my');

INSERT INTO Librarian
VALUES (DEFAULT, 'Ryann', 60377813162, 'ryann@nnu.edu.my');
INSERT INTO Librarian
VALUES (DEFAULT, 'Kristy', 60378065909, 'kristy@nnu.edu.my');
INSERT INTO Librarian
VALUES (DEFAULT, 'Sonal', 60342960528, 'sonal@nnu.edu.my');
```

Result:

LIBRARIANID	LIBRARIANNAME	LIBRARIANPHONE	LIBRARIANEMAIL
1	Eleanora	60321481445	eleanora@nnu.edu.my
2	Judy	60379563868	judy@nnu.edu.my
3	Felicia	60321480666	felicia@nnu.edu.my
4	Brielle	60356368656	brielle@nnu.edu.my
5	Merton	60362529712	merton@nnu.edu.my
6	Ryann	60377813162	ryann@nnu.edu.my
7	Kristy	60378065909	kristy@nnu.edu.my
8	Sonal	60342960528	sonal@nnu.edu.my

Student

Brief:

```
INSERT INTO Student
VALUES (
         1221101149,
         'Chevonne',
         60136276361,
         '1221101149@student.nnu.edu.my'
       );
```

Data Insertion:

```
INSERT INTO Student
VALUES (1221101149, 'Chevonne', 60136276361, '1221101149@student.nnu.edu.my');
INSERT INTO Student
VALUES (1211105688, 'Francis', 60137729564, '1211105688@student.nnu.edu.my');
INSERT INTO Student
VALUES (1201108250, 'Jayendra', 60139133146, '1201108250@student.nnu.edu.my');
INSERT INTO Student
VALUES (1201103172, 'Callahan', 60113229132, '1201103172@student.nnu.edu.my');
INSERT INTO Student
VALUES (1221308718, 'Norton', 60134045363, '1221308718@student.nnu.edu.my');

INSERT INTO Student
VALUES (1221108364, 'Beau', 60132282888, '1221108364@student.nnu.edu.my');
INSERT INTO Student
VALUES (1221300498, 'Darla', 60137880018, '1221300498@student.nnu.edu.my');
INSERT INTO Student
VALUES (1221305203, 'Marinda', 60135161075, '1221305203@student.nnu.edu.my');
INSERT INTO Student
VALUES (1221301044, 'Sita', 60136251234, '1221304494@student.nnu.edu.my');
INSERT INTO Student
VALUES (1221303040, 'Glory', 60139173281, '1221303040@student.nnu.edu.my');

INSERT INTO Student
VALUES (1221208712, 'Ibrahim', 60132142438, '1221208712@student.nnu.edu.my');
INSERT INTO Student
VALUES (1221301222, 'Pravina', 60132693501, '1221301222@student.nnu.edu.my');
INSERT INTO Student
VALUES (1211302375, 'John', 60137782015, '1211302375@student.nnu.edu.my');
INSERT INTO Student
VALUES (1191102350, 'Anna', 60132693474, '1191102350@student.nnu.edu.my');
INSERT INTO Student
VALUES (1211304795, 'Shelia', 60137873880, '1211304795@student.nnu.edu.my');

INSERT INTO Student
VALUES (1201307993, 'Kasen', 60132694083, '1201307993@student.nnu.edu.my');
INSERT INTO Student
VALUES (1211301717, 'Mckayla', 60133324741, '1211301717@student.nnu.edu.my');
INSERT INTO Student
```

```
VALUES (1201308385, 'Kyra', 60134107119, '1201308385@student.nnu.edu.my');
INSERT INTO Student
VALUES (1211204068, 'Bayley', 60132274143, '1211204068@student.nnu.edu.my');
INSERT INTO Student
VALUES (1221305204, 'Ferne', 6018431443, '1221305204@student.nnu.edu.my');

INSERT INTO Student
VALUES (1221305279, 'Anisha', 60135122977, '1221305279@student.nnu.edu.my');
INSERT INTO Student
VALUES (1211105085, 'Aura', 60134106373, '1211105085@student.nnu.edu.my');
INSERT INTO Student
VALUES (1221304645, 'Cece', 60132162037, '1221304645@student.nnu.edu.my');
INSERT INTO Student
VALUES (1181206202, 'Pete', 60138075920, '1181206202@student.nnu.edu.my');
INSERT INTO Student
VALUES (1221405993, 'Connor', 60134252101, '1221405993@student.nnu.edu.my');

INSERT INTO Student
VALUES (1221407600, 'Danielle', 60134044271, '1221407600@student.nnu.edu.my');
INSERT INTO Student
VALUES (1211206183, 'Marjorie', 60137877175, '1211206183@student.nnu.edu.my');
INSERT INTO Student
VALUES (1211401726, 'Sammi', 60139735230, '1211401726@student.nnu.edu.my');
INSERT INTO Student
VALUES (1201202218, 'Lynn', 60137727664, '1201202218@student.nnu.edu.my');
INSERT INTO Student
VALUES (1201209869, 'Abhinav', 60132070528, '1201209869@student.nnu.edu.my');

INSERT INTO Student
VALUES (1181209676, 'Zhen', 60173533362, '1181209676@student.nnu.edu.my');
INSERT INTO Student
VALUES (1221209586, 'Corina', 60133345117, '1221209586@student.nnu.edu.my');
```

Result:

STUDENTID	STUDENTNAME	STUDENTPHONE	STUDENTEMAIL
1221101149	Chevonne	60136276361	1221101149@student.nnu.edu.my
1211105688	Francis	60137729564	1211105688@student.nnu.edu.my
1201108250	Jayendra	60139133146	1201108250@student.nnu.edu.my
1201103172	Callahan	60113229132	1201103172@student.nnu.edu.my
1221308718	Norton	60134045363	1221308718@student.nnu.edu.my
1221108364	Beau	60132282888	1221108364@student.nnu.edu.my
1221300498	Darla	60137880018	1221300498@student.nnu.edu.my
1221305203	Marinda	60135161075	1221305203@student.nnu.edu.my
1221301044	Sita	60136251234	1221304494@student.nnu.edu.my
1221303040	Glory	60139173281	1221303040@student.nnu.edu.my
1221208712	Ibrahim	60132142438	1221208712@student.nnu.edu.my
1221301222	Pravina	60132693501	1221301222@student.nnu.edu.my
1211302375	John	60137782015	1211302375@student.nnu.edu.my
1191102350	Anna	60132693474	1191102350@student.nnu.edu.my
1211304795	Shelia	60137873880	1211304795@student.nnu.edu.my
1201307993	Kasen	60132694083	1201307993@student.nnu.edu.my
1211301717	Mckayla	60133324741	1211301717@student.nnu.edu.my
1201308385	Kyra	60134107119	1201308385@student.nnu.edu.my
1211204068	Bayley	60132274143	1211204068@student.nnu.edu.my
1221305204	Ferne	6018431443	1221305204@student.nnu.edu.my
1221305279	Anisha	60135122977	1221305279@student.nnu.edu.my
1211105085	Aura	60134106373	1211105085@student.nnu.edu.my
1221304645	Cece	60132162037	1221304645@student.nnu.edu.my
1181206202	Pete	60138075920	1181206202@student.nnu.edu.my
1221405993	Connor	60134252101	1221405993@student.nnu.edu.my
1221407600	Danielle	60134044271	1221407600@student.nnu.edu.my
1211206183	Marjorie	60137877175	1211206183@student.nnu.edu.my
1211401726	Sammi	60139735230	1211401726@student.nnu.edu.my
1201202218	Lynn	60137727664	1201202218@student.nnu.edu.my
1201209869	Abhinav	60132070528	1201209869@student.nnu.edu.my
1181209676	Zhen	60173533362	1181209676@student.nnu.edu.my
1221209586	Corina	60133345117	1221209586@student.nnu.edu.my

Publisher

Brief:

```
INSERT INTO Publisher
VALUES (
         DEFAULT,
         'Bioscied',
         '34, 1st Floor, Jalan Tanjung SD 13/2, Taman Sri Damansara, 52200 Kuala Lumpur,
Malaysia',
         'enquiries@bioscied.co',
         60362580694
       );
```

Data Insertion:

```
INSERT INTO Publisher
VALUES (DEFAULT, 'Bioscied', '34, 1st Floor, Jalan Tanjung SD 13/2, Taman Sri Damansara,
52200 Kuala Lumpur, Malaysia', 'enquiries@bioscied.co', 60362580694);
INSERT INTO Publisher
VALUES (DEFAULT, 'Sunflower Prints Inc', '8, Level 19, The Heritage Tower, Jalan Pahang,
Setapak, 53000 Kuala Lumpur, Malaysia', 'hello@sunflowerprints.inc', 60378062263);
INSERT INTO Publisher
VALUES (DEFAULT, 'Prodata Inc', '2nd Floor, Blok 488D, One Stop Centre, Jalan Burma, 10350
Penang, Malaysia', 'enquiries@prodata.inc', 60362572061);
INSERT INTO Publisher
VALUES (DEFAULT, 'Eller Learning', 'Wisma Eller, Jalan Ampang, 50450 Kuala Lumpur,
Malaysia', 'hi@ellerlearning.com', 60342560095);
INSERT INTO Publisher
VALUES (DEFAULT, 'HCI Publishing', '2nd Floor, The Mall, Jalan Putra, 50350 Kuala Lumpur,
Malaysia', 'business@hcipublishing.com.my', 60356317722);

INSERT INTO Publisher
VALUES (DEFAULT, 'Cadence Media Group', 'Lot 6, Block 44, Lebuh Tiga, 90000 Sandakan,
Sabah, Malaysia', 'enquiries@cadence.com', 60341057849);
INSERT INTO Publisher
VALUES (DEFAULT, 'Heinemann Educational Books', '55, Jalan Raja Perempuan Mazwin, Taman
Rishah, 30100 Perak, Malaysia', 'hello@heinemann.com', 60351626217);
INSERT INTO Publisher
VALUES (DEFAULT, 'Joan Adams Learning', 'Taman Tayton View, 6th Mile, Off Jalan Cheras,
56000 Kuala Lumpur, Malaysia', 'hello@joanadamslearning.co', 60340427895);
INSERT INTO Publisher
VALUES (DEFAULT, 'Idaho Press-Tribune Inc', '11, Seri Geriang, 35900 Hulu Bernam, Selangor,
Malaysia', 'enquiries@idahopt.inc', 60326930469);

INSERT INTO Publisher
VALUES (DEFAULT, 'Hickory Hills Publishing Co', 'Sungei Wang Plaza, Jalan Bukit Bintang,
55100 Kuala Lumpur, Malaysia', 'hi@hickoryhills.pub', 60387364511);
INSERT INTO Publisher
VALUES (DEFAULT, 'Woracle Publishing Inc', '367, Jalan Kuantan-Kemaman, Kampung Cherating
Lama, 26080 Kuantan, Malaysia', 'enquiries@woracle.com', 60333731617);
INSERT INTO Publisher
```

```
VALUES (DEFAULT, 'Sengage Learning', '3, Jalan 4/27F, Off Jalan Genting, Wangsa Maju, 53300
Klang, Selangor, Malaysia', 'hello@sengage.com', 60326978623);
INSERT INTO Publisher
VALUES (DEFAULT, 'Appleson Publishing', '1, Jalan Bakawali 11, Taman Johor Jaya, 81100
Johor, Malaysia', 'hello@appleson.pub', 60326932804);
```

Result:

PUBLISHERID	PUBLISHERNAME	PUBLISHERADDRESS	PUBLISHEREMAIL	PUBLISHERPHONE
1	Bioscied	34, 1st Floor, Jalan Tanjung SD 13/2, Taman Sri Damansara, 52200 Kuala Lumpur, Malaysia	enquiries@bioscied.co	60362588694
2	Sunflower Prints Inc	8, Level 19, The Heritage Tower, Jalan Pahang, Setapak, 53000 Kuala Lumpur, Malaysia	hello@sunflowerprints.inc	60378062263
3	Prodata Inc	2nd Floor, Blok 488D, One Stop Centre, Jalan Burma, 10350 Penang, Malaysia	enquiries@prodata.inc	60362572061
4	Eller Learning	Wisma Eller, Jalan Ampang, 50450 Kuala Lumpur, Malaysia	hi@ellerlearning.com	60342560095
5	HCI Publishing	2nd Floor, The Mall, Jalan Putra, 50350 Kuala Lumpur, Malaysia	business@hcipublishing.com.my	60356317722
6	Cadence Media Group	Lot 6, Block 44, Lebuh Tiga, 90000 Sandakan, Sabah, Malaysia	enquiries@cadence.com	60341057849
7	Heinemann Educational Books	55, Jalan Raja Permaisuri Mazwin, Taman Rishah, 30100 Perak, Malaysia	hello@heinemann.com	60351626217
8	Joan Adams Learning	Taman Tayton View, 6th Mile, Off Jalan Cheras, 56000 Kuala Lumpur, Malaysia	hello@joanadamslearning.co	60340427895
9	Idaho Press-Tribune Inc	11, Seri Gerlang, 35900 Hulu Bernam, Selangor, Malaysia	enquiries@idahopt.inc	60326930469
10	Hickory Hills Publishing Co	Sungei Wang Plaza, Jalan Bukit Bintang, 55100 Kuala Lumpur, Malaysia	hi@hickoryhills.pub	60387364511
11	Moracle Publishing Inc	367, Jalan Kuantan-Kemaman, Kampung Cherating Lama, 26080 Kuantan, Malaysia	enquiries@moracle.com	60333731617
12	Sengage Learning	3, Jalan 4/27F, Off Jalan Genting, Wangsa Maju, 53300 Klang, Selangor, Malaysia	hello@sengage.com	60326978623
13	Appleson Publishing	1, Jalan Bakawali 11, Taman Johor Jaya, 81100 Johor, Malaysia	hello@appleson.pub	60326932804

13 record(s) selected.

BookClass

Brief:

```
INSERT INTO BookClass
VALUES (
         000,
         'Computer Science, Information and General Works'
        );
```

Data Insertion:

```
INSERT INTO BookClass
VALUES (000, 'Computer Science, Information and General Works');
INSERT INTO BookClass
VALUES (100, 'Philosophy and Psychology');
INSERT INTO BookClass
VALUES (200, 'Religion');
INSERT INTO BookClass
VALUES (300, 'Social Sciences');
INSERT INTO BookClass
VALUES (400, 'Language');
INSERT INTO BookClass
VALUES (500, 'Science');
INSERT INTO BookClass
VALUES (600, 'Technology');
INSERT INTO BookClass
VALUES (700, 'Arts and Recreation');
INSERT INTO BookClass
VALUES (800, 'Literature');
INSERT INTO BookClass
VALUES (900, 'History and Geography');
```

Result:

```
BOOKCLASSID CLASS
----------- --------------------------------------------------
          0 Computer Science, Information and General Works
        100 Philosophy and Psychology
        200 Religion
        300 Social Sciences
        400 Language
        500 Science
        600 Technology
        700 Arts and Recreation
        800 Literature
        900 History and Geography

  10 record(s) selected.
```

Author

Brief:

```
INSERT INTO Author
VALUES (
         DEFAULT,
         'Annis Irma'
       );
```

Data Insertion:

```
INSERT INTO Author
VALUES (DEFAULT, 'Annis Irma');
INSERT INTO Author
VALUES (DEFAULT, 'Brigham Mack');
INSERT INTO Author
VALUES (DEFAULT, 'Sharifah Malandra');
INSERT INTO Author
VALUES (DEFAULT, 'Robina Ireland');
INSERT INTO Author
VALUES (DEFAULT, 'Chester Roger');

INSERT INTO Author
VALUES (DEFAULT, 'Aleah Everard');
INSERT INTO Author
VALUES (DEFAULT, 'Miranda Gail');
INSERT INTO Author
VALUES (DEFAULT, 'Torin Twyla');
INSERT INTO Author
VALUES (DEFAULT, 'Robena Herbert');
INSERT INTO Author
VALUES (DEFAULT, 'Clarissa Constance');

INSERT INTO Author
VALUES (DEFAULT, 'Genevieve Dhananjay');
INSERT INTO Author
VALUES (DEFAULT, 'Marni Chandana');
INSERT INTO Author
VALUES (DEFAULT, 'Braden Iris');
INSERT INTO Author
VALUES (DEFAULT, 'Jenifer Alfred');
INSERT INTO Author
VALUES (DEFAULT, 'Brock Michael');
```

Result:

```
AUTHORID    AUTHORNAME
----------- --------------------
          1 Annis Irma
          2 Brigham Mack
          3 Sharifah Malandra
          4 Robina Ireland
          5 Chester Roger
          6 Aleah Everard
          7 Miranda Gail
          8 Torin Twyla
          9 Robena Herbert
         10 Clarissa Constance
         11 Genevieve Dhananjay
         12 Marni Chandana
         13 Braden Iris
         14 Jenifer Alfred
         15 Brock Michael
```

Shelf

Brief:

```
INSERT INTO Shelf
VALUES (
         'S0101',
         1,
         1
        );
```

Data Insertion:

```
INSERT INTO Shelf
VALUES ('S0101', 1, 1);
INSERT INTO Shelf
VALUES ('S0102', 1, 2);
INSERT INTO Shelf
VALUES ('S0103', 1, 3);
INSERT INTO Shelf
VALUES ('S0104', 1, 4);
INSERT INTO Shelf
VALUES ('S0105', 1, 5);

INSERT INTO Shelf
VALUES ('S0201', 2, 1);
INSERT INTO Shelf
VALUES ('S0202', 2, 2);
INSERT INTO Shelf
VALUES ('S0203', 2, 3);
INSERT INTO Shelf
VALUES ('S0204', 2, 4);
INSERT INTO Shelf
VALUES ('S0205', 2, 5);

INSERT INTO Shelf
VALUES ('S0301', 3, 1);
INSERT INTO Shelf
VALUES ('S0302', 3, 2);
INSERT INTO Shelf
VALUES ('S0303', 3, 3);
INSERT INTO Shelf
VALUES ('S0304', 3, 4);
INSERT INTO Shelf
VALUES ('S0305', 3, 5);

INSERT INTO Shelf
VALUES ('S0401', 4, 1);
INSERT INTO Shelf
VALUES ('S0402', 4, 2);
INSERT INTO Shelf
VALUES ('S0403', 4, 3);
```

```sql
INSERT INTO Shelf
VALUES ('S0404', 4, 4);
INSERT INTO Shelf
VALUES ('S0405', 4, 5);

INSERT INTO Shelf
VALUES ('S0501', 5, 1);
INSERT INTO Shelf
VALUES ('S0502', 5, 2);
INSERT INTO Shelf
VALUES ('S0503', 5, 3);
INSERT INTO Shelf
VALUES ('S0504', 5, 4);
INSERT INTO Shelf
VALUES ('S0505', 5, 5);
```

Result:

```
SHELFCODE SHELFSECTION SHELFNUMBER
--------- ------------ -----------
S0101               1           1
S0102               1           2
S0103               1           3
S0104               1           4
S0105               1           5
S0201               2           1
S0202               2           2
S0203               2           3
S0204               2           4
S0205               2           5
S0301               3           1
S0302               3           2
S0303               3           3
S0304               3           4
S0305               3           5
S0401               4           1
S0402               4           2
S0403               4           3
S0404               4           4
S0405               4           5
S0501               5           1
S0502               5           2
S0503               5           3
S0504               5           4
S0505               5           5

  25 record(s) selected.
```

Book

Brief:

```
INSERT INTO Book
VALUES (
         DEFAULT,
         7,
         000,
         'S0101',
         9237412398742,
         'Computer Architecture and Organization',
         2020,
         315.30,
         DEFAULT
       );
```

Data Insertion:

```
INSERT INTO Book
VALUES (DEFAULT, 7, 000, 'S0101', 9237412398742, 'Computer Architecture and Organization',
2020, 315.30, DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 7, 000, 'S0101', 6372528593245, 'Discrete Mathematics with its
Applications', 2018, 345.50, DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 8, 000, 'S0102', 4674098503943, 'One Piece', 2002, 54.50, DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 7, 000, 'S0102', 8920384963823, 'Spirited Away', 1998, 323.56, 'LIB USE
ONLY');
INSERT INTO Book
VALUES (DEFAULT, 1, 100, 'S0103', 0923647812743, 'Atomic Habits', 2009, 34.12, DEFAULT);

INSERT INTO Book
VALUES (DEFAULT, 5, 000, 'S0102', 7826352647890, 'Running on the Roof of the World', 2007,
67.34, DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 5, 000, 'S0102', 8938462783764, 'Harry Potter and the Philosopher Stone',
2003, 63.99, DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 6, 500, 'S0204', 7835930459682, 'Chemistry', 1997, 234.80, DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 3, 600, 'S0205', 2946283759034, 'Fundamentals of Bioengineering', 2015,
134.50, DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 2, 000, 'S0101', 7892345626378, 'Calculus', 2017, 256.00, DEFAULT);

INSERT INTO Book
VALUES (DEFAULT, 4, 800, 'S0302', 7892345626378, 'Adventures of Molly Mood', 2013, 16.00,
'NOT AVAILABLE');
INSERT INTO Book
VALUES (DEFAULT, 4, 000, 'S0102', 7892345626378, 'Using Java 2', 2013, 782.00, DEFAULT);
```

```
INSERT INTO Book
VALUES (DEFAULT, 3, 200, 'S0205', 6374827469123, 'The Miracle of Theism', 2017, 87.50,
DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 3, 300, 'S0301', 6728475123904, 'Socialism across The World', 2015, 98.50,
DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 3, 400, 'S0401', 8937527364823, 'Japanese for Beginners', 2015, 201.88,
DEFAULT);

INSERT INTO Book
VALUES (DEFAULT, 3, 400, 'S0401', 6738429384721, 'Korean from Zero to Hero', 2022, 37.88,
DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 3, 900, 'S0504', 9592670266769, 'The Guns of August', 2021, 231.88,
DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 3, 800, 'S0502', 9874505376826, 'The Catcher in the Rye', 2020, 98.21,
DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 3, 700, 'S0404', 8074431102930, 'Steal like an Artist', 2019, 142.90,
DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 3, 600, 'S0401', 6684181963363, 'The Soul of a New Machine', 2018, 152.90,
DEFAULT);

INSERT INTO Book
VALUES (DEFAULT, 3, 800, 'S0502', 2946283759034, 'Wuthering Heights', 2013, 65.20,
DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 3, 900, 'S0503', 0486054388214, 'Earth and its Content', 2014, 242.30,
DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 3, 700, 'S0404', 9257310945337, 'Through the Lens of Saiful Nang', 2022,
189.00, DEFAULT);
INSERT INTO Book
VALUES (DEFAULT, 3, 700, 'S0405', 8674801196778, 'Performing Arts', 2016, 132.80, DEFAULT);
```

Result:

BOOKID	PUBLISHERID	BOOKCLASSID	SHELFCODE	SERIALNO	BOOKNAME	YEAROFPUBLISH	PRICE	BOOKCONDITION
1	7	0	S0101	9237412398742	Computer Architecture and Organization	2020	315.30	AVAILABLE
2	7	0	S0101	6372528593245	Discrete Mathematics with its Applications	2018	345.50	AVAILABLE
3	8	0	S0102	4674098503943	One Piece	2002	54.50	AVAILABLE
4	7	0	S0102	8920384963823	Spirited Away	1998	323.56	LIB USE ONLY
5	1	100	S0103	923647812743	Atomic Habits	2009	34.12	AVAILABLE
6	5	0	S0102	7826352647890	Running on the Roof of the World	2007	67.34	AVAILABLE
7	5	0	S0102	8938462783764	Harry Potter and the Philosopher Stone	2003	63.99	AVAILABLE
8	6	500	S0204	7835930459682	Chemistry	1997	234.80	AVAILABLE
9	3	600	S0205	2946283759034	Fundamentals of Bioengineering	2015	134.50	AVAILABLE
10	2	0	S0101	7892345626378	Calculus	2017	256.00	AVAILABLE
21	4	800	S0302	7892345626378	Adventures of Molly Mood	2013	16.00	NOT AVAILABLE
22	4	0	S0102	7892345626378	Using Java 2	2013	782.00	AVAILABLE
23	3	200	S0205	6374827469123	The Miracle of Theism	2017	87.50	AVAILABLE
24	3	300	S0301	2946283759034	Socialism across The World	2015	98.50	AVAILABLE
25	3	400	S0401	8937527364823	Japanese for Beginners	2015	201.88	AVAILABLE
26	3	400	S0401	6738429384721	Korean from Zero to Hero	2022	37.88	AVAILABLE
27	3	900	S0504	9592670266769	The Guns of August	2021	231.88	AVAILABLE
28	3	800	S0502	9874505376826	The Catcher in the Rye	2020	98.21	AVAILABLE
29	3	700	S0404	8074431102930	Steal like an Artist	2019	142.90	AVAILABLE
30	3	600	S0401	6684181963363	The Soul of a New Machine	2018	152.90	AVAILABLE
31	3	800	S0502	2946283759034	Wuthering Heights	2013	65.20	AVAILABLE
32	3	900	S0503	486054388214	Earth and its Content	2014	242.30	AVAILABLE
33	3	700	S0404	9257310945337	Through the Lens of Saiful Nang	2022	189.00	AVAILABLE
34	3	700	S0405	8674801196778	Performing Arts	2016	132.80	AVAILABLE

BookAuthor

Brief:

```
INSERT INTO BookAuthor
VALUES (
         1,
         1
       );
```

Data Insertion:

```
INSERT INTO BookAuthor
VALUES (1, 1);
INSERT INTO BookAuthor
VALUES (2, 6);
INSERT INTO BookAuthor
VALUES (3, 8);
INSERT INTO BookAuthor
VALUES (4, 7);
INSERT INTO BookAuthor
VALUES (5, 10);

INSERT INTO BookAuthor
VALUES (6, 3);
INSERT INTO BookAuthor
VALUES (7, 4);
INSERT INTO BookAuthor
VALUES (8, 1);
INSERT INTO BookAuthor
VALUES (9, 7);
INSERT INTO BookAuthor
VALUES (10, 7);

INSERT INTO BookAuthor
VALUES (9, 6);
INSERT INTO BookAuthor
VALUES (4, 1);
INSERT INTO BookAuthor
VALUES (4, 2);
INSERT INTO BookAuthor
VALUES (12, 9);
INSERT INTO BookAuthor
VALUES (12, 11);
INSERT INTO BookAuthor
VALUES (12, 12);
```

Result:

BOOKID	AUTHORID
1	1
2	6
3	8
4	7
5	10
6	3
7	4
8	1
9	7
10	7
22	9
22	11
22	12
22	28

Rental

Brief:

```
INSERT INTO Rental
VALUES (
         DEFAULT,
         1221305203,
         2,
         DATE('2022-11-08'),
         DATE('2022-11-29')
       );
```

Data Insertion:

```
INSERT INTO Rental
VALUES (DEFAULT, 1221305203, 2, DATE('2022-11-08'), DATE('2022-11-29'));
INSERT INTO Rental
VALUES (DEFAULT, 1221208712, 8, DATE('2022-11-13'), DATE('2022-12-04'));
INSERT INTO Rental
VALUES (DEFAULT, 1221101149, 1, DATE('2023-03-08'), DATE('2023-03-29'));
INSERT INTO Rental
VALUES (DEFAULT, 1211105688, 1, DATE('2023-03-08'), DATE('2023-03-29'));
INSERT INTO Rental
VALUES (DEFAULT, 1201209869, 5, DATE('2023-04-02'), DATE('2023-04-23'));

INSERT INTO Rental
VALUES (DEFAULT, 1211302375, 5, DATE('2023-05-18'), DATE('2023-06-08'));
INSERT INTO Rental
VALUES (DEFAULT, 1211401726, 4, DATE('2023-05-27'), DATE('2023-06-17'));
INSERT INTO Rental
VALUES (DEFAULT, 1221407600, 3, DATE('2023-05-29'), DATE('2023-06-19'));
```

```
INSERT INTO Rental
VALUES (DEFAULT, 1211206183, 7, DATE('2023-06-01'), DATE('2023-06-22'));
INSERT INTO Rental
VALUES (DEFAULT, 1221407600, 6, DATE('2023-06-06'), DATE('2023-06-27'));

INSERT INTO Rental
VALUES (DEFAULT, 1221208712, 8, DATE('2023-05-01'), DATE('2023-05-14'));
```

Result:

RENTALID	STUDENTID	LIBRARIANID	RENTALDATE	RETURNDATE
1	1221305203	2	11/08/2022	11/22/2022
2	1221208712	8	10/31/2022	11/13/2022
3	1221101149	1	03/08/2023	03/29/2023
4	1221101149	1	03/08/2023	03/29/2023
5	1201209869	5	02/25/2023	03/17/2023
6	1211302375	5	04/07/2023	04/28/2023
7	1211401726	4	05/11/2023	06/01/2023
8	1221407600	3	05/18/2023	06/08/2023
9	1211206183	7	05/21/2023	06/11/2023
10	1221405993	6	05/22/2023	06/12/2023
21	1221301044	6	06/21/0006	06/19/0006
22	1221405993	8	06/19/2023	06/26/2023
23	1221208712	8	05/01/2023	05/14/2023

RentalDetails

Brief:

```
INSERT INTO RentalDetails
VALUES (
          DEFAULT,
          1,
          1,
          DATE('2022-11-21'),
          DEFAULT
        );
```

Data Insertion:

```
INSERT INTO RentalDetails
VALUES (DEFAULT, 1, 1, DATE('2022-11-21'), DEFAULT);
INSERT INTO RentalDetails
VALUES (DEFAULT, 2, 5, DATE('2022-11-25'), DEFAULT);
INSERT INTO RentalDetails
VALUES (DEFAULT, 3, 8, DATE('2023-03-20'), DEFAULT);
INSERT INTO RentalDetails
VALUES (DEFAULT, 3, 9, DATE('2023-03-20'), DEFAULT);
INSERT INTO RentalDetails
VALUES (DEFAULT, 4, 10, DATE('2023-03-16'), DEFAULT);
```

```sql
INSERT INTO RentalDetails
VALUES (DEFAULT, 5, 8, DATE('2023-04-22'), DEFAULT);
INSERT INTO RentalDetails
VALUES (DEFAULT, 6, 9, DATE('2023-06-01'), DEFAULT);
INSERT INTO RentalDetails
VALUES (DEFAULT, 6, 7, DATE('2023-06-01'), DEFAULT);
INSERT INTO RentalDetails
VALUES (DEFAULT, 6, 5, DATE('2023-06-01'), DEFAULT);
INSERT INTO RentalDetails
VALUES (DEFAULT, 7, 3, DATE('2023-06-12'), DEFAULT);

INSERT INTO RENTALDETAILS
VALUES (DEFAULT, 8, 2, DATE('2023-06-12'), DEFAULT);
INSERT INTO RENTALDETAILS
VALUES (DEFAULT, 9, 4, DATE('2023-06-11'), DEFAULT);
INSERT INTO RENTALDETAILS
VALUES (DEFAULT, 10, 2, DATE('2023-06-12'), DEFAULT);
INSERT INTO RENTALDETAILS
VALUES (DEFAULT, 23, 3, DATE('2023-05-17'), 4.00);
```

Result:

RENTALDETAILSID	RENTALID	BOOKID	ACTUALRETURNDATE	CHARGE
1	1	1	11/21/2022	0.00
2	2	5	11/15/2022	0.00
3	3	8	03/29/2023	0.00
4	3	9	03/29/2023	0.00
5	4	10	03/22/2023	0.00
6	5	8	04/22/2023	0.00
7	6	9	06/01/2023	0.00
8	6	7	06/01/2023	0.00
9	6	5	06/01/2023	0.00
10	7	3	06/12/2023	0.00
45	23	3	05/17/2023	4.00

Reservation

Brief:

```
INSERT INTO Reservation
VALUES (
         DEFAULT,
         1221305203,
         3,
         1,
         DATE('2022-11-05'),
         DEFAULT
       );
```

Data Insertion:

```
INSERT INTO Reservation
VALUES (DEFAULT, 1221305203, 3, 1, DATE('2022-11-05'), DEFAULT);
INSERT INTO Reservation
VALUES (DEFAULT, 1221208712, 5, 4, DATE('2022-10-29'), DEFAULT);
INSERT INTO Reservation
VALUES (DEFAULT, 1221308718, 1, 5, DATE('2022-11-13'), DEFAULT);
INSERT INTO Reservation
VALUES (DEFAULT, 1221101149, 3, 8, DATE('2023-03-03'), DEFAULT);
INSERT INTO Reservation
VALUES (DEFAULT, 1181206202, 2, 9, DATE('2023-03-03'), DEFAULT);

INSERT INTO Reservation
VALUES (DEFAULT, 1211304795, 2, 7, DATE('2022-12-18'), DEFAULT);
INSERT INTO Reservation
VALUES (DEFAULT, 1191102350, 1, 6, DATE('2022-12-19'), DEFAULT);
INSERT INTO Reservation
VALUES (DEFAULT, 1201307993, 6, 2, DATE('2023-02-25'), DEFAULT);
INSERT INTO Reservation
VALUES (DEFAULT, 1201308385, 5, 3, DATE('2023-03-22'), DEFAULT);
INSERT INTO Reservation
VALUES (DEFAULT, 1211204068, 7, 10, DATE('2023-01-08'), DEFAULT);
```

Result:

```
RESERVEID   STUDENTID   LIBRARIANID BOOKID      RESERVEDATE
----------- ----------- ----------- ----------- -----------
          1  1221305203           3           1 11/05/2022
          2  1221208712           5           4 10/29/2022
          3  1221308718           1           5 11/13/2022
          4  1221101149           3           8 03/03/2023
          5  1181206202           2           9 03/03/2023
          6  1211304795           2           7 12/18/2022
          7  1191102350           1           6 12/19/2022
          8  1201307993           6           2 02/25/2023
          9  1201308385           5           3 03/22/2023
         10  1211204068           7          10 01/08/2023

  10 record(s) selected.
```

Data Manipulation with SQL

i. Aggregate functions:

<u>Count</u>

1. The purpose of this query is to count and retrieve the number of **different author(s)** that published books under a single **publisher**. The aggregate function 'COUNT' calculates the number of authors while the 'DISTINCT' keyword ensures that each author is counted only once, even if they have several books published by the same publisher. The 'JOIN' statements are used to establish the relationships between the tables. A set of rows, each representing a publisher and the number of authors associated with them, are returned as the result of this query. We can get information on how many different authors prefer to work with said publisher. This also allows users to see the diversity of authors published by each publisher.

```
SELECT PUBLISHER.PUBLISHERNAME,
       COUNT(DISTINCT AUTHOR.AUTHORID) AS NUMBEROFAUTHORS
FROM PUBLISHER
JOIN BOOK ON PUBLISHER.PUBLISHERID = BOOK.PUBLISHERID
JOIN BOOKAUTHOR ON BOOK.BOOKID = BOOKAUTHOR.BOOKID
JOIN AUTHOR ON BOOKAUTHOR.AUTHORID = AUTHOR.AUTHORID
GROUP BY PUBLISHER.PUBLISHERNAME;
```

	PUBLISHERNAME	NUMBEROFAUTHORS
1	Bioscied	1
2	Cadence Media Group	1
3	Eller Learning	3
4	HCI Publishing	2
5	Heinemann Educational Books	4
6	Joan Adams Learning	1
7	Prodata Inc	2
8	Sunflower Prints Inc	1

2. This query counts the total number of **books** published by each **publisher.** The 'JOIN' statement connects the 'Publisher' table with the 'Book' table by using 'PublisherID' as foreign key. Each row in the result reflects a publisher and the quantity of books they have released. The librarian can utilise this information to assess the library's partnerships with publishers, analyse the publishing trends and make informed decisions about collaboration.

```
SELECT PUBLISHER.PUBLISHERNAME,
       COUNT(BOOK.BOOKID) AS NUMBEROFBOOKS
FROM PUBLISHER
JOIN BOOK ON PUBLISHER.PUBLISHERID = BOOK.PUBLISHERID
GROUP BY PUBLISHER.PUBLISHERNAME;
```

	PUBLISHERNAME	NUMBEROFBOOKS
1	Bioscied	1
2	Cadence Media Group	1
3	Eller Learning	2
4	HCI Publishing	2
5	Heinemann Educational Books	3
6	Joan Adams Learning	1
7	Prodata Inc	13
8	Sunflower Prints Inc	1

3. This query displays the number of books written by a **single author** and the number of books with **multiple authors**. It gives a general overview of how many books in the library were written independently vs. written with other co-authors.

```
SELECT *
FROM
  (SELECT COUNT(*) AS SINGLEAUTHORBOOKS
   FROM
     (SELECT BOOKID
      FROM BOOKAUTHOR
      GROUP BY BOOKID
      HAVING COUNT(*) = 1)),

  (SELECT COUNT(*) AS MULTIPLEAUTHORBOOKS
   FROM
     (SELECT BOOKID
      FROM BOOKAUTHOR
      GROUP BY BOOKID
      HAVING COUNT(*) <> 1))
```

SINGLEAUTHORBOOKS	MULTIPLEAUTHORBOOKS
10	0

4. The reason for having this query is to count the number of books in each **book class** and specify the book class name. This query can be used by librarians to get an overview of the number of books available in each book class. The librarian can use this information to identify popular book classes, plan book acquisitions and assess the overall diversity of the library's book collections.

```
SELECT BOOKCLASS.BOOKCLASSID,
       CLASS,
       COUNT (*) AS TOTAL
FROM BOOK
LEFT JOIN BOOKCLASS ON BOOKCLASS.BOOKCLASSID = BOOK.BOOKCLASSID
GROUP BY BOOKCLASS.BOOKCLASSID,
         BOOKCLASS.CLASS
```

	123 BOOKCLASSID	ABC CLASS	123 TOTAL
1	0	Computer Science, Information and General Works	8
2	100	Philosophy and Psychology	1
3	200	Religion	1
4	300	Social Sciences	1
5	400	Language	2
6	500	Science	1
7	600	Technology	2
8	700	Arts and Recreation	3
9	800	Literature	3
10	900	History and Geography	2

Max

1. This query shows the **most expensive** book in each book class that is available for rental and **whether it is being borrowed**. The 'BCMax' subquery is joined with the 'Book' table based on the BookClassID and maximum price, ensuring that only the books with the highest price in each class are included. The 'CASE' statement determines the transaction ID based on the book condition. If the book condition is 'NOT AVAILABLE', it retrieves the rental ID of the latest rental where the book has not been returned yet. If the book condition is 'RESERVED', it retrieves the reserve ID of the latest reservation for the book. For any other book conditions, it sets the transaction ID as 'NULL'. This query allows librarians to track the availability of the most expensive book in each book class in the library.

```
SELECT BCMAX.CLASS,
       B.BOOKNAME,
       B.PRICE,
       B.BOOKCONDITION,
       CASE
           WHEN B.BOOKCONDITION = 'NOT AVAILABLE' THEN
                  (SELECT R.RENTALID
                   FROM RENTAL R
                   INNER JOIN RENTALDETAILS RD ON R.RENTALID = RD.RENTALID
                   WHERE RD.BOOKID = B.BOOKID
                     AND R.RETURNDATE IS NULL
                   ORDER BY R.RENTALDATE DESC
                   LIMIT 1)
           WHEN B.BOOKCONDITION = 'RESERVED' THEN
                  (SELECT RES.RESERVEID
                   FROM RESERVATION RES
                   WHERE RES.BOOKID = B.BOOKID
                   ORDER BY RES.RESERVEDATE DESC
                   LIMIT 1)
           ELSE NULL
       END AS TRANSACTIONID
FROM
  (SELECT BC.BOOKCLASSID,
          BC.CLASS,
          MAX(B.PRICE) AS MAXPRICE
   FROM BOOKCLASS BC
   JOIN BOOK B ON BC.BOOKCLASSID = B.BOOKCLASSID
   GROUP BY BC.BOOKCLASSID,
            BC.CLASS) AS BCMAX
JOIN BOOK B ON BCMAX.BOOKCLASSID = B.BOOKCLASSID
AND BCMAX.MAXPRICE = B.PRICE
ORDER BY BCMAX.CLASS;
```

CLASS	BOOKNAME	PRICE	BOOKCONDITION	TRANSACTIONID
Computer Science, Information and General Wor	Discrete Mathematics with its Applications	345.5	AVAILABLE	[NULL]
Philosophy and Psychology	Atomic Habits	34.12	AVAILABLE	[NULL]
Science	Chemistry	234.8	AVAILABLE	[NULL]
Technology	Fundamentals of Bioengineering	134.5	AVAILABLE	[NULL]

2. The purpose of this query is to show the **list** of books that have been **borrowed the most** regardless of category and show the details of these books. The 'DENSE_RANK()' function assigns a rank to each book based on the descending order of the count of rentals. The outer query joins the T1 subquery with the 'BOOK' table based on the book ID to retrieve detailed information about the books. This query provides librarians with valuable insights into the most rented books in the library. The librarian can use this information to understand the popularity of certain books, manage book inventory and ensure their availability for library users.

```
SELECT T1.RANK,
       T1.TIMESRENTED,
       T1.BOOKID,
       BOOKNAME,
       BOOKCLASSID,
       SHELFCODE,
       YEAROFPUBLISH,
       BOOKCONDITION
FROM
  (SELECT DENSE_RANK() OVER (
                              ORDER BY (COUNT(BOOKID)) DESC) AS RANK,
                             COUNT(BOOKID) AS TIMESRENTED,
                             BOOKID
   FROM RENTALDETAILS
   GROUP BY BOOKID) AS T1
LEFT JOIN BOOK ON T1.BOOKID = BOOK.BOOKID
WHERE RANK <= 3
ORDER BY RANK
```

RANK	TIMESRENTED	BOOKID	BOOKNAME	BOOKCLASSID	SHELFCODE	YEAROFPUBLISH	BOOK
1	2	3	One Piece	0	S0102	2,002	AVAILABL
1	2	5	Atomic Habits	100	S0103	2,009	AVAILABL
1	2	8	Chemistry	500	S0204	1,997	AVAILABL
1	2	9	Fundamentals of Bioengineering	600	S0205	2,015	AVAILABL
2	1	1	Computer Architecture and Organizatior	0	S0101	2,020	AVAILABL
2	1	7	Harry Potter and the Philosopher Stone	0	S0102	2,003	AVAILABL
2	1	10	Calculus	0	S0101	2,017	AVAILABL

Min

1. The reason for having this query is to select the **category** which has the mininum number of books among all other categories. In this case, the 'HAVING' clause helps to select the groups where the count of 'BookID' is equal to the minimum 'BookCount' obtained from the subquery. The librarians can weigh their decision on whether it would like to purchase more books for these specific categories. Additionally, this query can help librarians to identify the potential gaps in the collection of books and take appropriate actions to ensure a well-rounded selection of books for library users.

```
SELECT BC.BookClassID,
       BC.Class,
       COUNT(B.BookID) AS NumBooks
FROM BookClass BC
LEFT JOIN Book B ON BC.BookClassID = B.BookClassID
GROUP BY BC.BookClassID,
         BC.Class
HAVING COUNT(B.BookID) =
  (SELECT MIN(BookCount)
   FROM
     (SELECT COUNT(B.BookID) AS BookCount
      FROM BookClass BC
      LEFT JOIN Book B ON BC.BookClassID = B.BookClassID
      GROUP BY BC.BookClassID) AS BookCounts);
```

123 BOOKCLASSID	ABC CLASS	123 NUMBOOKS
100	Philosophy and Psychology	1
200	Religion	1
300	Social Sciences	1
500	Science	1

2. This query aims to find the shelf(s) with the minimum number of books and retrieve their details. The 'CREATE OR REPLACE VARIABLE' statement declares a variable named 'MINBOOKS' of type integer. It will be used to store the minimum number of books among all shelves. After this query, librarians can quickly find emptiest shelves to store books in order to maintain an organised and balanced distribution of books across different shelves.

```
CREATE OR REPLACE VARIABLE MINBOOKS INTEGER

SET MINBOOKS =
  (SELECT MIN(BOOKCOUNT)
   FROM
     (SELECT COUNT(*) BOOKCOUNT
      FROM SHELF
      GROUP BY SHELFCODE))


SELECT SHELFCODE,
       TOTALBOOKS,
       SHELFSECTION,
       SHELFNUMBER
FROM
  (SELECT SHELFCODE,
          COUNT(SHELFCODE) AS TOTALBOOKS,
          SHELFSECTION,
          SHELFNUMBER
   FROM
     (SELECT BOOK.SHELFCODE,
             SHELFSECTION,
             SHELFNUMBER
      FROM SHELF
      JOIN BOOK ON BOOK.SHELFCODE = SHELF.SHELFCODE) AS T1
   GROUP BY SHELFCODE,
            SHELFSECTION,
            SHELFNUMBER)
WHERE TOTALBOOKS = MINBOOKS
```

<u>Avg</u>

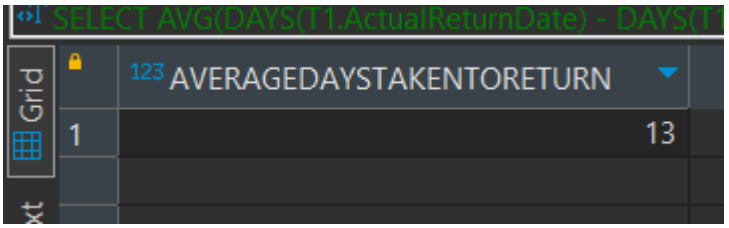
1. The purpose of this query is to calculate the average number of days taken by **each student** to **return a book.** The function DAYS() is used to calculate the number of days between two dates. The 'WHERE' clause 'WHERE ActualReturnDate IS NOT NULL' filters out rentals where the actual return date is not provided, considering only rentals that have been returned. By executing this query, the librarian can analyse the data and identify patterns related to student rental behaviour.

```
SELECT T1.STUDENTID,
       AVG(DAYS(T1.ACTUALRETURNDATE) - DAYS(T1.RENTALDATE)) AS
AVERAGEDAYSTAKENTORETURN
FROM
  (SELECT STUDENT.STUDENTID,
          RENTALDETAILS.ACTUALRETURNDATE,
          RENTAL.RENTALDATE
   FROM RENTAL,
        STUDENT,
        RENTALDETAILS
   WHERE RENTAL.STUDENTID = STUDENT.STUDENTID
     AND RENTALDETAILS.RENTALID = RENTAL.RENTALID) AS T1
WHERE ACTUALRETURNDATE IS NOT NULL
GROUP BY STUDENTID;
```

	STUDENTID	AVERAGEDAYSTAKENTORETURN
1	1,201,209,869	20
2	1,211,105,688	8
3	1,211,206,183	10
4	1,211,302,375	14
5	1,221,101,149	12
6	1,221,208,712	12
7	1,221,305,203	13
8	1,221,407,600	14

2. This query calculates the average number of days taken to **return a book** for **all students.** The function DAYS() is used to calculate the number of days between two dates. This query helps the librarians in understanding the general trend of rental return behaviour in the library. The librarian can monitor the average return time and assess if any improvements are needed to encourage timely returns.

```
SELECT AVG(DAYS(T1.ACTUALRETURNDATE) - DAYS(T1.RENTALDATE)) AS
AVERAGEDAYSTAKENTORETURN
FROM
  (SELECT STUDENT.STUDENTID,
          RENTALDETAILS.ACTUALRETURNDATE,
          RENTAL.RENTALDATE
   FROM RENTAL,
        STUDENT,
        RENTALDETAILS
   WHERE RENTAL.STUDENTID = STUDENT.STUDENTID
     AND RENTALDETAILS.RENTALID = RENTAL.RENTALID) AS T1
WHERE ACTUALRETURNDATE IS NOT NULL;
```



3. The reason for having this query is to calculate the average **number of books** borrowed by students. This query helps the librarian in understanding the reading habits and book borrowing patterns of the student population.

```
SELECT AVG(BORROWEDBOOKS) AS AVERAGEBOOKSREAD
FROM
  (SELECT STUDENTID,
          COUNT(*) AS BORROWEDBOOKS
   FROM
     (SELECT RENTAL.STUDENTID,
             RENTALDETAILS.RENTALDETAILSID
      FROM RENTALDETAILS,
           RENTAL
      WHERE RENTALDETAILS.RENTALID = RENTAL.RENTALID)
   GROUP BY STUDENTID);
```

123 AVERAGEBOOKSREAD
1

4. This query calculates the average **number of books** a **librarian handles** in a month. 'EXTRACT(MONTH FROM R.RentalDate)' is a function that extracts the month component from the 'RentalDate' column in the 'Rental' table. It is used to group the rental details by month. By executing this query, the librarian can assess the average workload of each librarian in terms of book handling. It provides insights into the distribution of book handling tasks across librarians and can help in evaluating the efficiency of library staff.

```
SELECT AVG(BOOKSHANDLED) AS AVERAGEBOOKSHANDLED
FROM
  (SELECT COUNT(*) AS BOOKSHANDLED
   FROM RENTALDETAILS RD
   JOIN RENTAL R ON RD.RENTALID = R.RENTALID
   GROUP BY EXTRACT(MONTH
                    FROM R.RENTALDATE),
            R.LIBRARIANID) AS MONTHLYBOOKSHANDLED;
```

123 AVERAGEBOOKSHANDLED
1

Sum

1. The purpose of this query is to sum up the **total penalties** of students who has not returned their books. Librarians can use this query to generate a report or overview of the penalty charges incurred by students for their late returns or other rental violations.

```
SELECT R.STUDENTID,
       RD.ACTUALRETURNDATE,
       SUM(RD.CHARGE) AS TOTALPENALTY
FROM RENTALDETAILS RD
JOIN RENTAL R ON RD.RENTALID = R.RENTALID
GROUP BY R.STUDENTID,
         RD.ACTUALRETURNDATE;
```

STUDENTID	ACTUALRETURNDATE	TOTALPENALTY
1,201,209,869	2023-04-22	0
1,211,302,375	2023-06-01	0
1,211,401,726	2023-06-12	0
1,221,101,149	2023-03-22	0
1,221,101,149	2023-03-29	0
1,221,208,712	2022-11-15	0
1,221,208,712	2023-05-17	4
1,221,305,203	2022-11-21	0

2. This query calculates the **total penalty** of **all books** for a specific month. This can be used and compared against the total penalty received for that month to understand how much out of the total late return fee that students pay.

```
CREATE OR REPLACE VARIABLE STARTDATERANGE DATE;
CREATE OR REPLACE VARIABLE ENDDATERANGE DATE;

SET STARTDATERANGE = '2023-06-01';
SET ENDDATERANGE = '2023-06-30';

SELECT SUM(RentalDetails.Charge) AS TotalPenalty
FROM RentalDetails,
     Rental
WHERE RentalDetails.RentalID = Rental.RentalID
  AND Rental.RentalDate >= STARTDATERANGE
  AND Rental.RentalDate <= ENDDATERANGE;
```

	123 TOTALPENALTY
1	[NULL]

ii. Query with a Group By and Having Clause

1. This query is designed to identify instances where a student has rented the same book multiple times. It retrieves data on the rental activity of students who have rented the same books more than once.

```
SELECT STUDENTID,
       STUDENTNAME,
       BOOKID,
       COUNT(*) AS TOTALTIMESRENTED
FROM
  (SELECT RENTAL.STUDENTID,
          STUDENTNAME,
          BOOKID
   FROM RENTALDETAILS,
        STUDENT,
        RENTAL
   WHERE RENTAL.RENTALID = RENTALDETAILS.RENTALID
     AND STUDENT.STUDENTID = RENTAL.STUDENTID)
GROUP BY STUDENTID,
         STUDENTNAME,
         BOOKID
HAVING COUNT(*) > 1;
```

123 STUDENTID	ABC STUDENTNAME	123 BOOKID	123 TOTALTIMESRENTED
1,221,407,600	Danielle	2	2

2. The purpose of this query is to count the number of books in each **category.** Librarians can use this query to obtain a summary of the number of books available in each book class and this helps them in understanding the distribution of books across different classes and enables them to assess the overall inventory.

```
SELECT BOOKCLASSID,
       CLASS,
       COUNT(*) AS NUMBEROFBOOKS
FROM BOOKCLASS
GROUP BY BOOKCLASSID,
         CLASS
HAVING COUNT(*) > 0
```

123 BOOKCLASSID	ABC CLASS	123 NUMBEROFBOOKS
0	Computer Science, Information and General Works	1
100	Philosophy and Psychology	1
200	Religion	1
300	Social Sciences	1
400	Language	1
500	Science	1
600	Technology	1
700	Arts and Recreation	1
800	Literature	1
900	History and Geography	1

3. This query retrieves information about the **total number of rented books** for each student. Librarians can use the data to monitor the popularity of different books among students and make informed decisions regarding book acquisitions.

```
SELECT T1.STUDENTID,
       T1.STUDENTNAME,
       COUNT(DISTINCT T1.BOOKID) AS TotalRentedBooks
FROM
  (SELECT STUDENT.STUDENTID,
          STUDENT.STUDENTNAME,
          RENTALDETAILS.RENTALDETAILSID,
          RENTAL.RENTALID,
          RENTALDETAILS.BOOKID
   FROM RENTAL,
        STUDENT,
        RENTALDETAILS
   WHERE RENTAL.STUDENTID = STUDENT.STUDENTID
     AND RENTALDETAILS.RENTALID = RENTAL.RENTALID) AS T1
GROUP BY T1.StudentID,
         T1.StudentName
```

123 STUDENTID	ABC STUDENTNAME	123 TOTALRENTEDBOOKS
1,221,101,149	Chevonne	2
1,211,105,688	Francis	1
1,221,305,203	Marinda	1
1,221,208,712	Ibrahim	1
1,211,302,375	John	3
1,221,407,600	Danielle	1
1,211,206,183	Marjorie	1
1,211,401,726	Sammi	1
1,201,209,869	Abhinav	1

4. The purpose of this query is for each student who has borrowed a book that is **not yet returned**, list down the **number of books** that they have borrowed

```
SELECT S.StudentID, S.StudentName, COUNT(RD.BookID) AS BorrowedBooks
FROM Student S
JOIN Rental R ON S.StudentID = R.StudentID
JOIN RentalDetails RD ON R.RentalID = RD.RentalID
WHERE RD.ActualReturnDate > R.ReturnDate
GROUP BY S.StudentID, S.StudentName
HAVING COUNT(RD.BookID) > 0;
```

STUDENTID	STUDENTNAME	BORROWEDBOOKS

iii. Triggers

1. This query is to **calculate** the **due date** of book return automatically. When a new rental is inserted into the 'RENTAL' table, the trigger will automatically calculate the return date as the rental date plus 21 days. This eliminates the need for manual entry or calculation of return dates for each rental, reducing the chances of errors.

```
CREATE TRIGGER SET_RETURNDATE AFTER
INSERT ON RENTAL
FOR EACH ROW MODE DB2SQL
UPDATE RENTAL
SET RETURNDATE =
  (SELECT rentaldate + 21 DAY
   FROM sysibm.sysdummy1);
```

SELECT * FROM RENTAL | Enter a SQL expression to filter results (use Ctrl+Space)

	RENTALID	STUDENTID	LIBRARIANID	RENTALDATE	RETURNDATE
13	13	1,201,108,250	4	2023-05-01	2023-05-22

2. The reason for having this query is after a student **returns** a book, calculate the total penalty for late return (if applicable) and set the book condition to BILLED. Otherwise, set the book condition to 'AVAILABLE'.

```
CREATE OR REPLACE TRIGGER RUNRETURNBOOK
        BEFORE
        UPDATE OF ACTUALRETURNDATE ON RENTALDETAILS
        REFERENCING NEW AS N
        FOR EACH ROW MODE DB2SQL
        BEGIN
                DECLARE RETURNDATE DATE;
                DECLARE CURRENTDATE DATE;

                SELECT RENTAL.RETURNDATE INTO RETURNDATE
                FROM RENTAL
                WHERE RENTAL.RENTALID = N.RENTALID;

                IF (N.ACTUALRETURNDATE > RETURNDATE) THEN
                        SET N.CHARGE = ((N.ACTUALRETURNDATE - RETURNDATE) * 2);
                        UPDATE BOOK
                        SET bookcondition = 'BILLED'
                        WHERE BOOK.BOOKID = N.BOOKID;
                END IF;

                IF (N.ACTUALRETURNDATE < RETURNDATE) THEN
                        SET N.CHARGE = 0;
                        UPDATE BOOK
                        SET bookcondition = 'AVAILABLE'
```

```
                    WHERE BOOK.BOOKID = N.BOOKID;
            END IF;
        END
```

	RENTALDETAILSID	RENTALID	BOOKID	ACTUALRETURNDATE	CHARGE
1	13	10	2	2023-06-30	6

3. This query ensures the ActualReturnDate should be larger than RentalDate. This trigger is used to enforce a validation rule on the 'ACTUALRETURNDATE' column of the 'RENTALDETAILS' table. If the return date is earlier than the rental date, the trigger raises an exception, preventing the update and displaying an error message. Librarians can rely on this trigger to ensure that return dates are not mistakenly set before the corresponding rental dates.

```
CREATE OR REPLACE TRIGGER RETURNDATEVALIDATIONTRIGGER
       BEFORE UPDATE OF ACTUALRETURNDATE ON RENTALDETAILS
       REFERENCING NEW AS N
       FOR EACH ROW MODE DB2SQL

       BEGIN
             DECLARE DATESTART DATE;

             SELECT RENTAL.RENTALDATE INTO DATESTART
             FROM RENTAL
             WHERE RENTAL.RENTALID = N.RENTALID;

             IF (N.ACTUALRETURNDATE < DATESTART) THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Return date must not be larger than rental
date.';
             END IF;
       END
```

```
--test update
UPDATE RENTALDETAILS SET ACTUALRETURNDATE = '2023-06-02' WHERE RENTALDETAILSID = 13
```

RENTALDETAILS 1 | RENTAL 2

Enter a SQL expression to filter results (use Ctrl+Space)

SQL Error [45000]: Application raised error or warning with diagnostic text: "Return date must not be larger than rental date.".. SQLCODE=-438, SQLSTATE=45000, DRIVER=4.31.10

Details >>

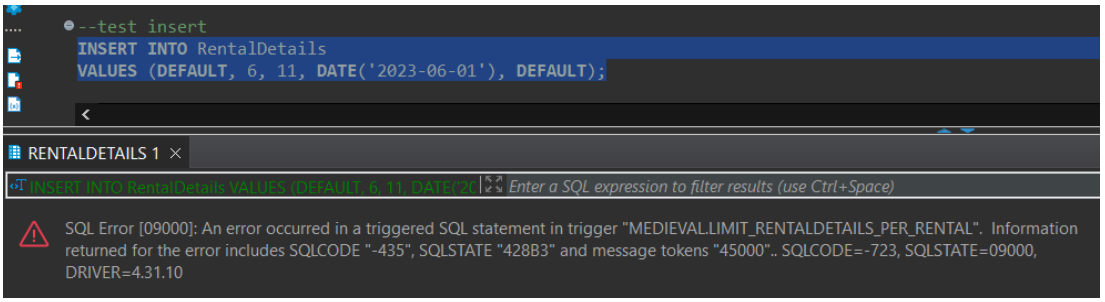
4. The provided code is a trigger designed to enforce a maximum limit of three RentalDetailsID entries for each RentalID in the RentalDetails table. The trigger is defined with the name "limit_rentaldetails_per_rental" and is set to execute before an insertion occurs on the RentalDetails table. It is configured to reference the newly inserted row using the "NEW" keyword, and the trigger logic is enclosed within the

"BEGIN ATOMIC" and "END" keywords. The "DECLARE" statement creates a variable called "count_rentaldetails" to store the count of RentalDetails records associated with the RentalID of the new row. The "SET" statement assigns the count of RentalDetails records to the "count_rentaldetails" variable. If the count is greater than or equal to three, the "SIGNAL" statement raises an exception with SQLSTATE '45000' and sets the MESSAGE_TEXT to 'A student may only borrow at most 3 books at a time'.

```
CREATE TRIGGER limit_rentaldetails_per_rental
BEFORE INSERT ON RentalDetails
REFERENCING NEW AS N
FOR EACH ROW
MODE DB2SQL
BEGIN ATOMIC
    DECLARE count_rentaldetails INTEGER;

    -- Count the number of RentalDetails already associated with the RentalID
    SET count_rentaldetails = (
        SELECT COUNT(*)
        FROM RentalDetails
        WHERE RentalID = N.RentalID
    );

    -- If there are already 3 RentalDetails for this RentalID, raise an exception
    IF count_rentaldetails >= 3 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'A student may only borrow at most 3 books at a time';
    END IF;
END
```



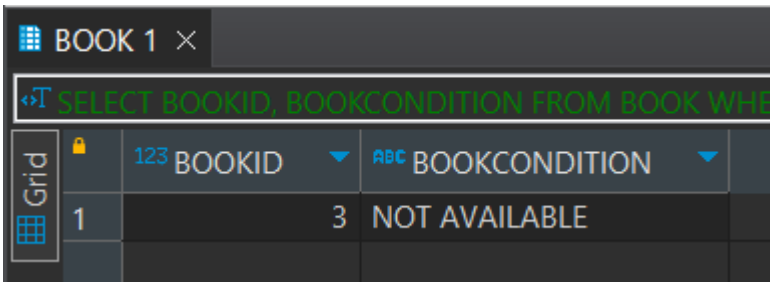
5. The script creates a trigger named "INSERTRENTALDETAILSTRIGGER" that fires after an insert operation on the "RENTALDETAILS" table. It uses the CREATE OR REPLACE TRIGGER statement to create an alias for the newly inserted row, the FOR EACH ROW MODE DB2SQL specifies that the trigger should execute for each row affected by the insert operation, the DECLARE statement to declare a variable to store the previous book condition, the SELECT statement to retrieve the previous book condition, the IF statement to check if the previous book condition is 'AVAILABLE', and the SIGNAL statement to raise an exception with SQLSTATE '45000'. The purpose of this SQL script is to create a trigger that updates the book

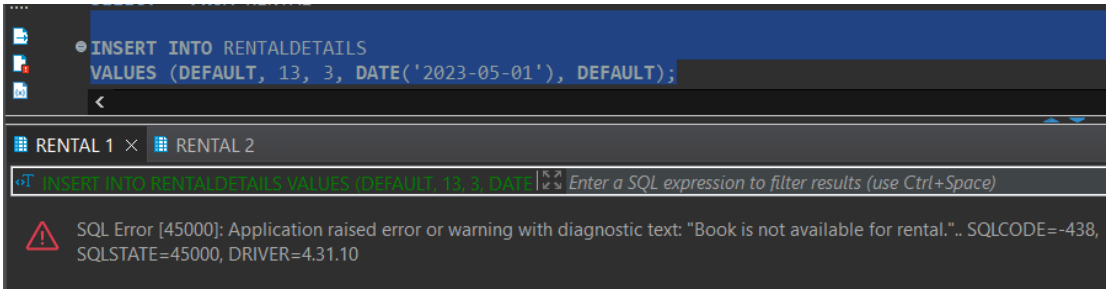
condition to 'NOT AVAILABLE' for the corresponding book when a new rental detail is inserted into the "RENTALDETAILS" table. This trigger helps maintain book availability, prevent illegal rentals, and maintain data integrity. It also includes an exception handling mechanism to indicate that the book is not available for rental.

```
CREATE OR REPLACE TRIGGER INSERTRENTALDETAILSTRIGGER
      AFTER INSERT ON RENTALDETAILS
      REFERENCING NEW AS N
      FOR EACH ROW MODE DB2SQL
      BEGIN
            DECLARE PREVIOUSBOOKCONDITION VARCHAR(15);

            SELECT BOOK.BOOKCONDITION INTO PREVIOUSBOOKCONDITION
            FROM BOOK
            WHERE BOOK.BOOKID = N.BOOKID;

            IF (PREVIOUSBOOKCONDITION = 'AVAILABLE') THEN
                UPDATE BOOK
                SET bookcondition = 'NOT AVAILABLE'
                WHERE BOOK.BOOKID = N.BOOKID;
            ELSE SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Book is not available for rental.';
            END IF;
      END
```





iv. Stored Procedure

1. The InsertRental stored procedure is designed to handle the insertion of rental records into a database. It accepts two input parameters: p_bookID and p_rentalDate. Two variables are declared: rental_ID and conditionOfBook. A SELECT statement is used to retrieve the BookCondition value from the Book table for the specified book ID. If the book is available, an INSERT statement is executed to add a new rental record into the Rental table.The IDENTITY_VAL_LOCAL() function is used to retrieve the last generated identity value (rental ID). Finally, an INSERT statement is executed to add a new rental details record into the RentalDetails table.

```
CREATE OR REPLACE PROCEDURE InsertRental (IN p_bookID INT, IN p_rentalDate DATE)
BEGIN
      DECLARE rental_ID INT DEFAULT 0;
      DECLARE conditionOfBook VARCHAR(15);

      SELECT BookCondition INTO conditionOfBook
      FROM Book
      WHERE BookID = p_bookID;

      IF conditionOfBook <> 'AVAILABLE' THEN
         SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'The selected book is not available for rental.';
      ELSE
         INSERT INTO Rental (RentalDate)
         VALUES (p_rentalDate);

         SET rental_ID = IDENTITY_VAL_LOCAL();

         INSERT INTO RentalDetails (RentalID, BookID)
         VALUES (rental_ID, p_bookID);
      END IF;
END

--Call the procedure by using--
CALL InsertRental(3, '2023-06-15');
```

Statistics 1

Name	Value
Updated Rows	-1
	CALL InsertRental(3, '2023-06-15')
Start time	Mon Jun 19 17:21:42 SGT 2023
Finish time	Mon Jun 19 17:21:42 SGT 2023

SELECT * FROM RENTAL | Enter a SQL expression to filter results (use Ctrl+Space)

Grid

	RENTALID	STUDENTID	LIBRARIANID	RENTALDATE	RETURNDATE
12	12	[NULL]	[NULL]	2023-06-15	2023-07-06

SELECT * FROM RENTALDETAILS | Enter a SQL expression to filter results (use Ctrl+Space)

Grid

	RENTALDETAILSID	RENTALID	BOOKID	ACTUALRETURNDATE	CHARGE
14	15	12	3	[NULL]	[NULL]

v. View

1. The provided SQL script creates a view called "StudentTransactions" that displays the details of a particular student and all their past transactions, including the list of books they have rented or reserved and their corresponding details. The CREATE VIEW statement creates a view named "StudentTransactions" that contains the result set of the SELECT query. The SET statement assigns a particular student ID to the variable "SELECTEDSTUDENTID". The SELECT statement retrieves the following columns from multiple tables: S.StudentID, S.StudentName, R.RentalID, R.RentalDate, NULL AS ReserveID, NULL AS ReserveDate, 'Rental' AS TransactionType, B.BookID, B.BookName, B.YearOfPublish. The "StudentTransactions" view provides a comprehensive view of a student's past transactions, including both rentals and reservations. It includes the student's ID, name, and relevant details of each transaction, allowing library staff to efficiently manage and track a student's transaction history. Customization based on the selected student allows library staff to view transaction details for individual students as needed.

```
CREATE OR REPLACE VARIABLE SELECTEDSTUDENTID INT;

SET SELECTEDSTUDENTID = 1221305203;

CREATE VIEW StudentTransactions AS
SELECT S.StudentID,
       S.StudentName,
       R.RentalID,
       R.RentalDate,
       NULL AS ReserveID,
       NULL AS ReserveDate,
       'Rental' AS TransactionType,
       B.BookID,
       B.BookName,
       B.YearOfPublish
FROM Student S
LEFT JOIN Rental R ON S.StudentID = R.StudentID
LEFT JOIN RentalDetails RD ON R.RentalID = RD.RentalID
LEFT JOIN Book B ON RD.BookID = B.BookID
WHERE S.StudentID = SELECTEDSTUDENTID
UNION ALL
SELECT S.StudentID,
       S.StudentName,
       NULL AS RentalID,
       NULL AS RentalDate,
       RES.ReserveID,
       RES.ReserveDate,
       'Reservation' AS TransactionType,
       B.BookID,
       B.BookName,
       B.YearOfPublish
```

```
FROM Student S
LEFT JOIN Reservation RES ON S.StudentID = RES.StudentID
LEFT JOIN Book B ON RES.BookID = B.BookID
WHERE S.StudentID = SELECTEDSTUDENTID;

--Display the view by using--
SELECT *
FROM StudentTransactions;
```

	STUDENTID	STUDENTNAME	RENTALID	RENTALDATE	RESERVEID	RESERVEDATE	TRANSACTIONTYPE	BOOKID	BOOKNAME
1	1,221,305,203	Marinda	[NULL]	[NULL]	[NULL]	[NULL]	Reservation	[NULL]	[NULL]
2	1,221,305,203	Marinda	1	2022-11-08	[NULL]	[NULL]	Rental	1	Computer Architecture and Organization

2. The script creates a view named "StudentsWithDueCharges" to display students who have due charges or haven't returned their books. It retrieves the following columns from multiple tables: s.StudentID, s.StudentName, rd.Charge, and r.ReturnDate. JOIN clauses join the "Student," "Rental," and "RentalDetails" tables. WHERE clause filters rows based on two conditions: r.ReturnDate CURRENT_DATE and rd.ActualReturnDate IS NULL. This SQL script creates a view that displays the details of students who have due charges or haven't returned their books, along with the corresponding charge amount and due date. It helps identify students with overdue books, track charges, track and remind students about due dates, and efficiently manage and track overdue books and associated charges.

```
CREATE VIEW StudentsWithDueCharges AS
SELECT s.StudentID,
       s.StudentName,
       rd.Charge,
       r.ReturnDate
FROM Student s
JOIN Rental r ON s.StudentID = r.StudentID
JOIN RentalDetails rd ON r.RentalID = rd.RentalID
WHERE r.ReturnDate < CURRENT_DATE
  AND rd.ActualReturnDate IS NULL;

SELECT *FROM StudentsWithDueCharges;
```

STUDENTID	STUDENTNAME	CHARGE	RETURNDATE

3. The provided SQL script creates a view called "BooksToBeReturned" that lists all books that need to be returned. The CREATE VIEW statement creates a view named "BooksToBeReturned" that contains the result set of the SELECT query. The WHERE clause filters the rows based on two conditions: ReturnDate IS NOT NULL and Current_Date >= ReturnDate. The SELECT statement following the view creation selects all rows from the "BooksToBeReturned" view to display the list of books that need to be returned. This script is useful for centralized list, automation, timely reminders, efficient book management, and enforcing timely returns. It provides a centralized and automated solution for tracking overdue books, enabling libraries to better manage their collections and borrower responsibilities.

```sql
CREATE VIEW BooksToBeReturned AS
SELECT *
FROM Rental
WHERE ReturnDate IS NOT NULL AND CURRENT_DATE >= ReturnDate;

SELECT * FROM BooksToBeReturned;
```

SELECT * FROM BooksToBeReturned | Enter a SQL expression to filter results (use Ctrl+Space)

	RENTALID	STUDENTID	LIBRARIANID	RENTALDATE	RETURNDATE
1	1	1,221,305,203	2	2022-11-08	2022-11-29
2	2	1,221,208,712	8	2022-11-13	2022-12-04
3	3	1,221,101,149	1	2023-03-08	2023-03-29
4	4	1,211,105,688	1	2023-03-08	2023-03-29
5	5	1,201,209,869	5	2023-04-02	2023-04-23
6	6	1,211,302,375	5	2023-05-18	2023-06-08
7	7	1,211,401,726	4	2023-05-27	2023-06-17
8	8	1,221,407,600	3	2023-05-29	2023-06-19
9	11	1,221,208,712	8	2023-05-01	2023-05-14

4. The provided SQL script retrieves a list of books written by a specific author and includes the associated publisher ID and name. It begins by declaring and initializing a variable named CHOSENAUTHOR with a value of 1. The SELECT statement retrieves the columns BOOKAUTHOR.BOOKID, BOOKNAME, and PUBLISHERNAME. The FROM clause specifies the tables used in the query and the LEFT JOIN statements connect the tables based on their relationships. The ON clauses define the join conditions and the WHERE clause filters the rows based on the condition BOOKAUTHOR.AUTHORID = CHOSENAUTHOR.The ORDER BY clause sorts the result set based on the BOOKAUTHOR.BOOKID column. This script is useful for author-specific book list, publisher information, flexibility, integration with book management systems, and valuable insights into the author's published works.

```
CREATE OR REPLACE VARIABLE CHOSENAUTHOR INTEGER;

SET CHOSENAUTHOR = 1;

CREATE VIEW ALLBOOKSUNDERAUTHOR AS
SELECT BOOKAUTHOR.BOOKID,
       BOOKNAME,
       PUBLISHERNAME
FROM BOOKAUTHOR
LEFT JOIN BOOK ON BOOK.BOOKID = BOOKAUTHOR.BOOKID
LEFT JOIN PUBLISHER ON PUBLISHER.PUBLISHERID = BOOK.PUBLISHERID
WHERE BOOKAUTHOR.AUTHORID = CHOSENAUTHOR;

SELECT * FROM ALLBOOKSUNDERAUTHOR;
```

SELECT * FROM ALLBOOKSUNDERAUTHOR | Enter a SQL expression to filter results (use Ctrl+Spac

	BOOKID	BOOKNAME	PUBLISHERNAME
1	8	Chemistry	Cadence Media Group
2	1	Computer Architecture and Organization	Heinemann Educational Books
3	4	Spirited Away	Heinemann Educational Books

5. The provided SQL script creates a view named "BookSearch" that allows users to search for books containing similar keywords. The CREATE VIEW statement creates a view with columns such as BookID, ShelfCode, BookName, YearOfPublish, AuthorID, AuthorName, and BookCondition. The FROM clause of the SELECT statement specifies the tables used in the query. The WHERE clause establishes the conditions for joining the tables. The LIKE operator with the % wildcard is used to match any occurrence of the keyword in the BookName column.The SELECT * statement retrieves all columns from the BookSearch view. The BookSearch view simplifies the process of searching for books based on keywords, allowing users to query the view and retrieve relevant book details without manually constructing complex SQL queries. It also allows for flexible searching, time-saving, and improved user experience. By creating a view and encapsulating the search logic, users can avoid repeating the search query construction each time they need to search for books with similar keywords. Overall, the BookSearch view enhances the book searching capabilities by allowing users to search for books that contain similar keywords, providing a user-friendly and efficient approach to retrieve relevant book details.

```sql
CREATE VIEW BookSearch AS
SELECT B.BookID,
       B.ShelfCode,
       B.BookName,
       B.YearOfPublish,
       BA.AuthorID,
       A.AuthorName,
       B.BookCondition
FROM Book B,
     Author A,
     BookAuthor BA
WHERE B.BookID = BA.BookID
  AND BA.AuthorID = A.AuthorID
  AND LOWER(B.BookName) LIKE '%the%';

--Display the view by using--
SELECT *
FROM BookSearch;
```

SELECT * FROM BookSearch | Enter a SQL expression to filter results (use Ctrl+Space)

	BOOKID	SHELFCODE	BOOKNAME	YEAROFPUBLISH	AUTHORID	AUTHORNAME	BOOKCONDITION
1	6	S0102	Running on the Roof of the World	2,007	3	Sharifah Malandra	AVAILABLE
2	7	S0102	Harry Potter and the Philosopher Stone	2,003	4	Robina Ireland	AVAILABLE
3	2	S0101	Discrete Mathematics with its Applications	2,018	6	Aleah Everard	AVAILABLE

vi. Subqueries / Nested Queries

1. This query retrieves the details of the publisher(s) with the most books in the database, including their contact information. It begins with a common table expression (CTE) named "PublisherBookCounts" and joins the "Publisher" and "Book" tables based on the PublisherID column. The COUNT() function is used to calculate the number of books associated with each publisher. The main SELECT statement selects the PublisherID, PublisherName, PublisherAddress, PublisherEmail, PublisherPhone, and NumberOfBooks from the "PublisherBookCounts" CTE. The WHERE clause filters the results to include only the publisher(s) with the highest number of books. This query is valuable for various reasons, such as recognition of publishers, contact information availability, decision-making and partnerships, and understanding and interaction with publishers who have made substantial contributions to the library's book collection.

```sql
WITH PublisherBookCounts AS
  (SELECT Publisher.PublisherID,
          Publisher.PublisherName,
          Publisher.PublisherAddress,
          Publisher.PublisherEmail,
          Publisher.PublisherPhone,
          COUNT(Book.BookID) AS NumberOfBooks
   FROM Publisher
   JOIN Book ON Publisher.PublisherID = Book.PublisherID
   GROUP BY Publisher.PublisherID,
            Publisher.PublisherName,
            Publisher.PublisherAddress,
            Publisher.PublisherEmail,
            Publisher.PublisherPhone)


SELECT PublisherBookCounts.PublisherID,
       PublisherBookCounts.PublisherName,
       PublisherBookCounts.PublisherAddress,
       PublisherBookCounts.PublisherEmail,
       PublisherBookCounts.PublisherPhone,
       PublisherBookCounts.NumberOfBooks
FROM PublisherBookCounts
WHERE PublisherBookCounts.NumberOfBooks =
    (SELECT MAX(NumberOfBooks)
     FROM PublisherBookCounts);
```

	PUBLISHERID	PUBLISHERNAME	PUBLISHERADDRESS	PUBLISHEREMAIL	PUBLISHERPHONE	NUMBEROFBOOKS
1	3	Prodata Inc	2nd Floor, Blok 488D, One Stop Centre, Jalan Burma, 10350 Penang, Malaysia	enquiries@prodata.inc	60,362,572,061	13

vii. At least 4 queries not covered in lecture

1. Count the total number of books of **each book condition** and add a column to sum **total number of books that aren't 'available'**. The provided query counts the total number of books for each book condition and adds a column to sum the total number of books that are not in the 'available' condition. It consists of a UNION of two SELECT statements that calculate the total number of books that are not in the 'available' condition and adds a row with the label 'SUM OF BOOKS THAT CANNOT BE RENTED'. The LIMIT 1 clause ensures that only one row is returned. The second SELECT statement retrieves the book condition and counts the total number of books for each condition using the COUNT() function. The UNION ALL operator combines the results of the two SELECT statements into a single result set. The purpose of this query is to obtain a comprehensive count of books for each book condition, including a separate row that sums up the total number of books that cannot be rented ('available' condition excluded).

```
(SELECT 'SUM OF BOOKS THAT CANNOT BE RENTED' AS BOOKCONDITION,
     (SELECT COUNT(*)
      FROM BOOK
      WHERE BOOKCONDITION <> 'AVAILABLE') AS TOTALBOOKS
   FROM BOOK
   WHERE BOOKCONDITION <> 'AVAILABLE'
   LIMIT 1)
UNION ALL
SELECT BOOKCONDITION,
       COUNT (BOOKCONDITION) AS TOTALBOOKS
FROM BOOK
GROUP BY BOOKCONDITION
```

	BOOKCONDITION	TOTALBOOKS
1	AVAILABLE	22
2	LIB USE ONLY	1
3	NOT AVAILABLE	1
4	SUM OF BOOKS THAT CANNOT BE RENTED	2

2. This query retrieves the most borrowed book in each category and displays its details. It starts by creating a common table expression (CTE) named "RankedBooks" which joins the "RentalDetails" table with the "Book" table using the "BookID" column and calculates the count of times each book has been rented in each category. The RANK() function is used with the PARTITION BY clause to rank the books within each category based on the number of times they have been rented. The main query selects columns from the "RankedBooks" CTE, including the book ID, book name, book class ID, shelf code, year of publish, book condition, and times rented. The WHERE clause filters the results based on the rank in each category and the ORDER BY clause sorts the results by book class ID.This query provides valuable insights into the popularity of books within different categories, allowing the library to make informed decisions regarding book acquisitions, restocking, and promoting popular books to attract more readers. Additionally, it assists in understanding reading preferences, enabling the library to curat

```sql
WITH RankedBooks AS
  (SELECT RentalDetails.BookID,
          Book.BookName,
          Book.BookClassID,
          Book.ShelfCode,
          Book.YearOfPublish,
          Book.BookCondition,
          COUNT(RentalDetails.BookID) AS TimesRented,
          RANK() OVER (PARTITION BY Book.BookClassID
                       ORDER BY COUNT(RentalDetails.BookID) DESC) AS RankInCategory
   FROM RentalDetails
   JOIN Book ON RentalDetails.BookID = Book.BookID
   GROUP BY RentalDetails.BookID,
            Book.BookName,
            Book.BookClassID,
            Book.ShelfCode,
            Book.YearOfPublish,
            Book.BookCondition)

SELECT RankedBooks.BookID,
       RankedBooks.BookName,
       RankedBooks.BookClassID,
       RankedBooks.ShelfCode,
       RankedBooks.YearOfPublish,
       RankedBooks.BookCondition,
       RankedBooks.TimesRented
FROM RankedBooks
WHERE RankedBooks.RankInCategory = 1
ORDER BY RankedBooks.BookClassID;
```

	BOOKID	BOOKNAME	BOOKCLASSID	SHELFCODE	YEAROFPUBLISH	BOOKCONDITION	TIMESRENTED
1	2	Discrete Mathematics with its Applications	0	S0101	2,018	AVAILABLE	2
2	5	Atomic Habits	100	S0103	2,009	AVAILABLE	2
3	8	Chemistry	500	S0204	1,997	AVAILABLE	2
4	9	Fundamentals of Bioengineering	600	S0205	2,015	AVAILABLE	2

3. This query retrieves a list of student details who have borrowed books the least amount of times or have never borrowed books. It also includes a message to promote reading based on the borrowing behavior. The query starts by selecting student details from the "Student" table and using a LEFT JOIN to join the "Student" table with the "Rental" table. The COUNT() function is used to count the number of rentals for each student and assigns it to the column alias "BorrowedBooks". The CASE statement is used to generate the "PromoteMessage" column. The query groups the results by student details and the HAVING clause is used to filter the results.

```
SELECT Student.StudentID,
       Student.StudentName,
       Student.StudentPhone,
       Student.StudentEmail,
       COALESCE(COUNT(Rental.StudentID), 0) AS BorrowedBooks,
       CASE
           WHEN COALESCE(COUNT(Rental.StudentID), 0) = 0 THEN 'Promote Reading'
           ELSE 'Encourage More Reading'
       END AS PromoteMessage
FROM Student
LEFT JOIN Rental ON Student.StudentID = Rental.StudentID
GROUP BY Student.StudentID,
         Student.StudentName,
         Student.StudentPhone,
         Student.StudentEmail
HAVING COALESCE(COUNT(Rental.StudentID), 0) =
  (SELECT MIN(BorrowedCount)
   FROM
     (SELECT COUNT(StudentID) AS BorrowedCount
      FROM Rental
      GROUP BY StudentID))
OR COUNT(Rental.StudentID) = 0;
```

	STUDENTID	STUDENTNAME	STUDENTPHONE	STUDENTEMAIL	BORROWEDBOOKS	PROMOTEMESSAGE
1	1,181,206,202	Pete	60,138,075,920	1181206202@student.nnu.edu.my	0	Promote Reading
2	1,181,209,676	Zhen	60,173,533,362	1181209676@student.nnu.edu.my	0	Promote Reading
3	1,191,102,350	Anna	60,132,693,474	1191102350@student.nnu.edu.my	0	Promote Reading
4	1,201,103,172	Callahan	60,113,229,132	1201103172@student.nnu.edu.my	0	Promote Reading
5	1,201,108,250	Jayendra	60,139,133,146	1201108250@student.nnu.edu.my	0	Promote Reading
6	1,201,202,218	Lynn	60,137,727,664	1201202218@student.nnu.edu.my	0	Promote Reading
7	1,201,209,869	Abhinav	60,132,070,528	1201209869@student.nnu.edu.my	1	Encourage More Reading
8	1,201,307,993	Kasen	60,132,694,083	1201307993@student.nnu.edu.my	0	Promote Reading
9	1,201,308,385	Kyra	60,134,107,119	1201308385@student.nnu.edu.my	0	Promote Reading
10	1,211,105,085	Aura	60,134,106,373	1211105085@student.nnu.edu.my	0	Promote Reading
11	1,211,105,688	Francis	60,137,729,564	1211105688@student.nnu.edu.my	1	Encourage More Reading
12	1,211,204,068	Bayley	60,132,274,143	1211204068@student.nnu.edu.my	0	Promote Reading
13	1,211,206,183	Marjorie	60,137,877,175	1211206183@student.nnu.edu.my	1	Encourage More Reading
14	1,211,301,717	Mckayla	60,133,324,741	1211301717@student.nnu.edu.my	0	Promote Reading
15	1,211,302,375	John	60,137,782,015	1211302375@student.nnu.edu.my	1	Encourage More Reading
16	1,211,304,795	Shelia	60,137,873,880	1211304795@student.nnu.edu.my	0	Promote Reading
17	1,211,401,726	Sammi	60,139,735,230	1211401726@student.nnu.edu.my	1	Encourage More Reading
18	1,221,101,149	Chevonne	60,136,276,361	1221101149@student.nnu.edu.my	1	Encourage More Reading
19	1,221,108,364	Beau	60,132,282,888	1221108364@student.nnu.edu.my	0	Promote Reading
20	1,221,209,586	Corina	60,133,345,117	1221209586@student.nnu.edu.my	0	Promote Reading
21	1,221,300,498	Darla	60,137,880,018	1221300498@student.nnu.edu.my	0	Promote Reading

4. The provided code is a SQL query that retrieves the details of books due for return within a specific date range. Two variables, STARTDATERANGE and ENDDATERANGE, are created and assigned the start and end dates of the desired date range. The SELECT statement retrieves various details related to the books that are due for return within the specified date range. The WHERE clause defines the relationships between the relevant tables and filters the results based on the return date.

```
CREATE OR REPLACE VARIABLE STARTDATERANGE DATE;
CREATE OR REPLACE VARIABLE ENDDATERANGE DATE;

SET STARTDATERANGE = '2023-06-01';
SET ENDDATERANGE = '2023-06-30';

SELECT RENTAL.rentalid,
       RENTAL.STUDENTID,
       STUDENTNAME,
       STUDENTPHONE,
       STUDENTEMAIL,
       RENTALDETAILSID,
       BOOK.BOOKID,
       BOOKNAME,
       RETURNDATE
FROM RENTAL,
     RENTALDETAILS,
     STUDENT,
     BOOK
WHERE RENTAL.RENTALID = RENTALDETAILS.RENTALID
  AND STUDENT.STUDENTID = RENTAL.STUDENTID
  AND RENTALDETAILS.BOOKID = BOOK.BOOKID
  AND RETURNDATE BETWEEN STARTDATERANGE AND ENDDATERANGE;
```

	RENTALID	STUDENTID	STUDENTNAME	STUDENTPHONE	STUDENTEMAIL	RENTALDETAILSID	BOOKID	BOOKNAME	RETURNDATE
1	6	1,211,302,375	John	60,137,782,015	1211302375@student.nnu.edu.my	9	5	Atomic Habits	2023-06-08
2	6	1,211,302,375	John	60,137,782,015	1211302375@student.nnu.edu.my	7	9	Fundamentals of Bioengineering	2023-06-08
3	6	1,211,302,375	John	60,137,782,015	1211302375@student.nnu.edu.my	8	7	Harry Potter and the Philosopher Stone	2023-06-08
4	8	1,221,407,600	Danielle	60,134,044,271	1221407600@student.nnu.edu.my	11	2	Discrete Mathematics with its Applicatic	2023-06-19
5	10	1,221,407,600	Danielle	60,134,044,271	1221407600@student.nnu.edu.my	13	2	Discrete Mathematics with its Applicatic	2023-06-27
6	9	1,211,206,183	Marjorie	60,137,877,175	1211206183@student.nnu.edu.my	12	4	Spirited Away	2023-06-22
7	7	1,211,401,726	Sammi	60,139,735,230	1211401726@student.nnu.edu.my	10	3	One Piece	2023-06-17

Conclusion

In conclusion, a database management system is essential for colleges, schools, and many more places these days. It is a very efficient tool in helping to store and organize data especially for a university library where tens of thousands of books may be stored and where students loan and reserve dozens of books on a daily basis. Using a database management system provides various benefits such as automated record keeping, efficient data retrieval, and enhanced security. Manual work can be heavily reduced with the help of a database management system. Queries can be used to obtain information for various research and informational purposes. Additionally, the system supports various use cases, including managing book records and tracking book rentals. Due to the large number of librarians reading and modifying data about the library books and its users, a database management system is an ideal solution to avoid various manual errors in the data management process. However, various improvements can still be done for this university library database to streamline the process of managing the library system data. In the future, the database can still be further customized to suit the unique needs of the university.

Contributions

1.	1211103146	Cheryl Gwee En Xin	100%
2.	1221303203	Toh Ee Lin	100%
3.	1221303636	Nabila Nadia binti Md Zaid	100%
4.	1221303972	Lee Jia Ying	100%