Library Database Management System
ECS740P
Group 9

Tai Ho Cheuk
Yusuf Sindi
Chengxuan Huang
Safiya Sharif

Table of Contents

Introduction

This report covers the process involved in implementing a relation schema by providing the code that was used to create tables, views, triggers, and insert and query data, using Oracle as the relational database management system. The implementation is based on a revised version of the normalized relational schema of the library system developed in coursework 1. Revisions of the schema are detailed and discussed. Constraints and triggers were used to provide integrity and robustness to the system, as well as to fulfil the requirements and functionality required by the system. This is complemented by an additional section on security. The code was run and tested on Oracle's Live SQL. The original relational schema is exhibited below (primary keys are underlined and foreign keys are in italics).

member(cardNo, *type*, firstName, lastName, suspendedStatus)
memberType(type, loanQuota)
emailAddress(email, *cardNo.*)
phoneNumber(phoneNo, *cardNo*)
loan(*cardNo*, *resourceId*, *copyId*, checkinDate, checkoutDate)
libraryStaff(*cardNo*., *role*)
libraryStaffRole(role, dbAccess)
resource(resourceId, resourceType, title, publisher, price, publicationYear, maxLoanPeriod, *floorNumber, shelfNumber*)
shelf(shelfNumber, *classNumber*)
floor(classNumber, floor)
class(classNumber, className)
resourceCopy(*resourceId*, copyId)
resourceCreator(*resourceId*, creator)

Relational Schema

To ensure a better implementation of our relational schema, some necessary changes were made. In this section, all the alterations to our relational schema will be discussed.
Several changes were made to two loan tables - ‘Loan’ and ‘Historical Loan’. First of all, we have renamed the two tables to ‘currentLoan’ and ‘pastLoan’ respectively, because these new names are more intuitive when writing the SQL code. In each loan table, a new primary key is given because we have decided to add an attribute named ‘currentLoanId’ to the Current Loan table; and an attribute named ‘pastLoanId’ to the Past Loan table. These two loan ID attributes will both be auto-increment numbers which serve as the only primary key to their respective table. The three attributes ‘cardNo’, ‘resourceId’, and ‘copyId’ are now foreign keys which reference the Member table and Resource Copy table.
Additionally, a ‘fines’ attribute is added to the currentLoan table. In our original design, fines are not stored in any table because all fines are derived value. However, considering fines could be valuable information to the library management by enabling them to identify persistent offenders, we have decided to store fines in the currentLoan table.
Another major change we have made is that a new table named paymentHistory is added to the relational schema. The purpose of the new table is to keep track of the historical fine payments. The primary key of this table is “pastLoanId”, which is also a foreign key that references the past loan table. The only other attribute of this table is fines. The data stored in this table will be automatically inserted whenever an overdue resource item is being checked in or a fine is cleared for an item previously checked in. That is, when a member returns an item that has a fine or clear a fine for a previously returned item, a trigger will be fired to move the data from the currentLoan table to the Past Loan table and to the Payment History table. The mechanism of the trigger will be discussed in detail in the Trigger section.
Changes were also made to the shelf and class table. Firstly, the attribute classNumber is renamed classCode because instead of using random number to identify a class, we have decided to have a 4-character code to represent each unique class. The reason of the renaming is a code that is made up with English letters is easier to remember. It is because resources of a class can only appear on one floor, hence, if we know the class code we will know the floor number at which a class locates. It is therefore the floor table and class table from our old schema are combined to form a new class table. The only difference between the new and old class table is the attribute ‘floor’ is added to the new class table. The rationale is both class name and floor are attributes that are functionally dependent on the primary key – class code.
Two changes were also made to the Resources table. Firstly, a non-key attribute ‘ISBN’ is added to the table. This column will store the unique International Standard Book Number of any book that has it. NULL will be entered for resources that do not have an ISBN. The second change made is a foreign key – ‘classCode’ – was added to the table to replace the ‘floorNumber’ attribute. The rationale of replacing floor number with class code is that all resources of the same class are stored on one and only one floor.
The name of the table Library Staff Library and Library Staff Role have been shortened to ‘libStaff’ and ‘libRole’ respectively. It is because shorter table names are preferable when writing SQL code. Moreover, in the libRole table, we have divided the attribute ‘Database Access’ into two new attributes – readAccess and writeAccess. Both new attributes are binary valued which means the values can only be either ‘0’ or ‘1’. The two new attributes are used to indicate whether a specific role has the write access or read access to the database system. Further details will be discussed in the Table Creation section.

Additionally, two minor alterations to note are firstly, the Resource table has been renamed Resources to avoid confusion because the word ‘resource’ is a keyword in Oracle SQL. Secondly, the new column named ‘countryCode’ is added to the phoneNumber table considering there may be member who only has a phone number from his home country. Country code and phone number together form a composite primary key for the phoneNumber table.
After all the changes made, now we have a final version of relational schema. All the SQL codes discussed in later sections will be based on this version of relational schema.

member (cardNo, *mType*, firstName, lastName, suspendedStatus)
memberType (mType, loanQuota)
emailAddress (Email, *cardNo)*
phoneNumber (countryCode, phoneNo, *cardNo)*
currentLoan (currentLoanId, *cardNo, resourceId, copyId*, checkoutDate, checkinDate, fines)
pastLoan (pastLoanId, *cardNo*, *resourceId, copyId*, checkoutDate, checkinDate)
libStaff (*cardNo*, *role*)
libRole (role, readAccess, writeAccess)
resources (resourceId, ISBN, resourceType, title, publisher, price, publicationYear, maxLoanPeriod, *shelfNumber, classCode*)
class (classCode, className, floorNumber)
shelf (shelfNumber, classCode)
resourceCopy (*resourceId*, copyId)
resourceCreator (*resourceId*, creator)
paymentHistory (*pastLoanId*, *fines*)

Table Creation

This section will present and discuss all the SQL statements required to create the tables of our relational schema.

memberType
The mType attribute is the primary key of this table, which determines the maximum number of resource copies a member can borrow. The latter is stored as loanQuota with a numeric data type. It will be given a not null constraint as it is important to determine how many days a member is allowed to burrow a book.

```
CREATE TABLE memberType (
   mType VARCHAR2(40),
   loanQuota NUMBER(2) NOT NULL,
   PRIMARY KEY(mType)
);
```

member
The cardNo attribute indicates card number and is the primary key that uniquely identifies a member. In our design, we chose to make it a string made of 10 numbers which may begin with one or more zeros. To that end, cardNo is set as a CHAR type with a fixed length of 10 and a check in put in place to ensure all its characters are numbers. For lack of Boolean data type in Oracle, we stipulated that suspendedStatus can only take values 0 or 1 to indicate whether a person is suspended or not. Possible values of member types are given in the memberType table, so mType is set as a foreign key referencing that table. Two additional check constraints are devised to ensure that names begin with capital letters.

```
CREATE TABLE member (
   cardNo CHAR(10),
   mType VARCHAR2(40) NOT NULL,
   firstName VARCHAR2(40) NOT NULL,
   lastName VARCHAR2(40) NOT NULL ,
   suspendedStatus NUMBER(1) DEFAULT (0),
   PRIMARY KEY(cardNo), /* cardNo is primary key for table */
   FOREIGN KEY (mType)
      REFERENCES memberType(mType), /* Foreign key references memberType */
   CHECK (INITCAP(firstName) = firstName),
   CHECK (INITCAP(lastName) = lastName),
   CHECK (TRIM (TRANSLATE (TRANSLATE (cardNo,' ','a'),'0123456789',' ')) IS
NULL), /*check to see if the member card number is a string of numbers*/
   CHECK (suspendedStatus IN (0,1))
);
```

emailAddress
Since we decided to allow members to provide multiple e-mail addresses, we created a separate table for them. As e-mail addresses are always unique, they have been set as the primary key for the table. The cardNo attribute is a foreign key referencing the member table and as e-mails cannot exist without card numbers, an on delete cascade has been set as well. Two check conditions have been added to verify the format of e-mail entries.

```
CREATE TABLE emailAddress (
   email VARCHAR2(40),
   cardNo CHAR(10) NOT NULL,
   PRIMARY KEY(email), /* Email is primary key for table */
   FOREIGN KEY (cardNo)
      REFERENCES member(cardNo)
      ON DELETE CASCADE, /* Foreign key references member table */
   CHECK (email LIKE '%@%'), /* Check email contains @ */
   CHECK (email LIKE '%.%') /* Check email contains */
);
```

phoneNumber
As per the same logic for email addresses, we created a separate table to store members' phone numbers. An adjustment worth noting is that we decided to introduce country codes as an additional attribute to cater to the needs of international members. Both country codes and phone numbers have been set as the composite primary to uniquely identify each tuple in the table. For the same reason given for the emailAddress table, an ON DELETE CASCADE constraint has been put in place for the cardNo foreign key. Finally, since phone numbers can begin with one or more zeros, it is necessary to place a check constraint that ensures phone numbers are strings made entirely of numbers.

```
 CREATE TABLE phoneNumber (

   countryCode NUMBER(3),

   phoneNo VARCHAR(20),

   cardNo CHAR(10) NOT NULL,

   PRIMARY KEY (countryCode, phoneNo), /*Primary key for table*/

   FOREIGN KEY (cardNo)

      REFERENCES member(cardNo)

      ON DELETE CASCADE, /* Foreign key references member table */

   CHECK (TRIM (TRANSLATE (TRANSLATE (phoneNo,' ','a'), '0123456789', ' ')) IS
NULL)  /*check if the phone number is a string of numbers*/

);
```

libRole

The column role is given a primary key constraint in the table libRole, which means that each value in this column uniquely identifies each row. By default, role must be unique and not null.

As Oracle does not have a Boolean datatype, readAccess and writeAccess, have a check constraint that declares that the value in the column must be set to either 0 or 1, with 0 indicating that the corresponding library role does not have access to the database, and 1 indicating that it does. The two attributes also have a DEFAULT constraint, where each new row that is added is set to 0 unless otherwise specified.

```
CREATE TABLE libRole (
   role VARCHAR2 (40) PRIMARY KEY,
   readAccess NUMBER (1) DEFAULT 0 CHECK (readAccess IN (1,0)),
   writeAccess NUMBER (1) DEFAULT 0 CHECK (writeAccess IN (1,0))
);
```

libStaff
The libStaff table comprises two columns, where cardNo is the primary key and role is a foreign key referencing the libRole table. Both keys are constrained hence they must not be null and must be unique.

```
CREATE TABLE libStaff (
   cardNo CHAR(10) PRIMARY KEY,
   role VARCHAR2 (40) NOT NULL,
   FOREIGN KEY (cardNo)
      REFERENCES member(cardNo)
      ON DELETE CASCADE,
   FOREIGN KEY (role)
      REFERENCES libRole(role)
);
```

class
Because both className and floorNumber depend on classCode, classCode will have a primary key constraint. Thus, all class codes must be unique and not null. The datatype of class code is constrained to five characters using a check constraint. className is constrained by not null. Instead of assigning a random class number to each class name, we have decided to adopt a 5-character code format to uniquely identify each class name, for increased readability. The string datatype also imposes a character limit of 40 on className. The other attribute, floor, has a check constraint imposed on it, which ensures the value entered must be either 0, 1 or 2, as the library only has three floors. The single-digit numbers are used to distinguish the three levels of the library, where 0 stands for ground floor, 1 for first floor, and 2 for second floor. The constraint is complemented by the datatype allowing only single digit numbers for this attribute.

```
CREATE TABLE class (
   classCode CHAR(4),
   className VARCHAR(40) NOT NULL,
   floorNumber NUMBER(1) NOT NULL, /* Must be 0, 1 or 2 */
   PRIMARY KEY (classCode), /* classCode is the primary key */
   CHECK (LENGTH(classCode) = 4), /* classCode must be four-character string */
   CHECK (floorNumber IN (0, 1, 2))
);
```

shelf
In the shelf table, shelfNumber and classCode will make up the primary key, so all values must not be null and must be unique. The datatype for shelfNumber imposes a limit of 4 numbers, since there are repeating shelves on each floor and we never expect to have more that 9,999 shelves per floor. The attribute classCode is a foreign key referencing classCode in the Class table. The constraint of the foreign key helps enforce the referential integrity by limiting classCode to only the values that already exist in the class table. By default, the foreign key has the no action option so if the corresponding class code tries to get deleted in the class table, it will raise an error.

```
CREATE TABLE shelf (
   shelfNumber NUMBER(4),
   classCode CHAR(4),
   PRIMARY KEY (shelfNumber, classCode),
   FOREIGN KEY (classCode)
      REFERENCES class(classCode)
);
```

resources
The resources table was given the primary key resourceId. This was the preferred method instead of making ISBN the primary key as we acknowledged that not all resources will have an ISBN. The ISBN attribute has the datatype varchar2, since some ISBN's might start with a 0, and giving it a number constraint will eliminate any leading zeroes. The ISBN has a check constraint that ensures that each ISBN entered is either 10 or 13 characters, as they were 10 digits before 2007, but are now 13. There will be other descriptive attributes in the resources table, such as resourceType, publisher, price and publication year. The resources table will also have a maxLoanPeriod which specifies how long the resource can be checked out for without the member incurring any fines and will have a datatype that imposes 2 digits as the library never expects to allow a member to loan the book for more than 99 days (about 3 and a half months). The maxLoanPeriod can also be set to –1 which will indicate that the resource is no longer available. While we could have instead introduced a new attribute that more clearly depicts this property, we chose this approach to avoid redunancy as most resources will not be discarded. The shelfNumber and classCode are also given a foreign key constraint that references the shelf table. The shelf with its associated class code must exist to be present in the resources table. By default, the foreign key has the no action option which means that if the shelf number and class code tries to get deleted, it will raise an error. Finally, the attributes resourceType, title, maxLoanPeriod, shelfNumber, and classCode were given the not null constraint as these are all important information to keep about each resource.

```
CREATE TABLE resources (
   resourceId NUMBER (7),
   ISBN VARCHAR2(13),
   resourceType VARCHAR2(20) NOT NULL,
   title VARCHAR2(60) NOT NULL,
   publisher VARCHAR2(20),
   price NUMBER(6, 2),
   publicationYear DATE,
   maxLoanPeriod NUMBER(2) NOT NULL, /* Maximum loan period in days */
   shelfNumber NUMBER(4) NOT NULL,
   classCode CHAR(4) NOT NULL,
   PRIMARY KEY (resourceId),
   FOREIGN KEY (shelfNumber, classCode)
      REFERENCES shelf(shelfNumber, classCode),
   CHECK (LENGTH(ISBN) IN (10, 13))
);
```

resourceCopy
The resourceCopy table exists to minimize redundancy in the resources table while allowing to keep record of different copies. The attributes resourceId and copyId together will make up the primary key, which means they must be unique and not null. copyId will be give the number datatype and will impose a limit of two digits, as we do not expect the library told hold more than 99 copies of a particular resource. resourceId will be given a foreign key constraint referencing resourceId in the resources table, and thus must exist in the resources table to exist in the resourceCopy table. The foreign key also has the on delete cascade option which means that if the resourceId is deleted in the resources table, it will also be deleted in the resourceCopy table.

```
CREATE TABLE resourceCopy (
   resourceId NUMBER(7),
   copyId NUMBER(2) NOT NULL ,
   PRIMARY KEY (resourceId, copyId),
   FOREIGN KEY (resourceId)
      REFERENCES resources(resourceId)
);
```

resourceCreator
The resource creator table was also created to minimize redundancy as the creator attribute is a repeating group attribute. The primary keys in this table are both resourceId and creator. resourceId is given a foreign key constraint referencing resoruceId in the resources table. The foreign key also has the on delete cascade option which means that if the resourceId is deleted in the resources table, it will also be deleted in the resourceCreator table. However, in most cases, a trigger will be used to delete a resource to maintain information about the resources. Further information will be provided in the trigger section.

```
CREATE TABLE resourceCreator (
   resourceId NUMBER(7),
   creator VARCHAR2(40),
   PRIMARY KEY (resourceId, creator),
   FOREIGN KEY (resourceId)
      REFERENCES resources(resourceId)
      ON DELETE CASCADE
);
```

currentLoan
While cardNo, resourceId, copyId, and checkoutDate were all canidate keys, it was decided that the loan table would have a unique primary key with a number that autoincrements every time a row is created. While there is no auto_increment feature available in Oracle, Oracle 12c database introduced the ability to define an identity clause, which allowed to simulate the auto_increment behaviour by inserting null as the currentLoanId value. This was done to

avoid having a long composite primary key, and additionally allows the library to keep track of the order of the loans that have been checked out. cardNo is given a foreign key constraint referencing the primary key cardNo in the member table and resourceId and copyId are given a foreign key constraint referencing resourceId and copyId in the resourceCopy table. No options under the foreign keys are specified, and thus, the no action option is enforced which means an error will be raised if either the associated resource or member is deleted. checkinDate and checkoutDate consist of the data type timestamp to determine the exact time and date a loan has been checked out. The attribute fines is a derived attribute and will store the amount a member owes on a loan, which will increase by 1 every day the resource is overdue. Finally, cardNo, resourceId, copyId, checkoutDate and fines have the constraint not null as they are important information to know if ever there was a need to refer to the data pertaining to a particular loan. Furthermore, it would enable the library to keep track of which copies of a resource are not in the library at a particular instance in time, which member is currently holding them, and how long they have had that resource.

```
CREATE TABLE currentLoan (
   currentLoanId NUMBER GENERATED by default on null as IDENTITY,
   cardNo CHAR(10) NOT NULL,
   resourceId NUMBER(7) NOT NULL,
   copyId NUMBER (2) NOT NULL,
   checkoutDate TIMESTAMP NOT NULL,
   checkinDate TIMESTAMP,
   fines NUMBER(6,2) DEFAULT (0) NOT NULL, /* The amount a member owes which
will be converted to GBP */
   PRIMARY KEY (currentLoanId),
   FOREIGN KEY (cardno)
      REFERENCES member(cardNo),
   FOREIGN KEY (resourceId, copyId) REFERENCES resourceCopy(resourceId, copyId),
   CHECK(checkinDate > checkoutDate)
);
```

pastLoan
The pastLoan table will have its own unique ID, pastLoanId, for the same reasons stated in the currentLoan section. Conclusively, pastLoanId will have a primary key constraint, where it cannot be null and must be unique. It will have the same ID as the currentLoan ID when the data is transferred into that table. It will also have all the same attributes as the currentLoan table with the same foreign-key constraints, except for fines which will be recorded in the paymentTable. Having a pastLoan table will save database resources as a check to see if fines should be incremented will not have to be executed for every loan that has existed.

```
CREATE TABLE pastLoan (
   pastLoanId NUMBER(9) PRIMARY KEY,
   cardNo CHAR(10),
   resourceId NUMBER(7),
   copyId NUMBER(2),
   checkoutDate TIMESTAMP NOT NULL,
   checkinDate TIMESTAMP,
   FOREIGN KEY (cardno)
      REFERENCES member(cardNo),
   FOREIGN KEY (resourceId, copyId) REFERENCES resourceCopy(resourceId, copyId),
   CHECK(checkinDate > checkoutDate)
);
```

paymentHistory
The paymentHistory table will keep a record of the amount paid for fines on a loan. The attribute pastLoanId will have a primary key constraint (which will mean it will have to be unique and not null) and will also have a foreign key constraint referencing the pastLoanId in the pastLoan table. By default, since not otherwise specified, the foreign key is given the no action option which means if the associated row in the pastLoan table tries to get deleted, the database system will raise an error. The reference will enable the library system to keep track of all the information of the loan that was associated with a payment, such as the member, the resource, and when the loan was checked in and checked out. It will also increase efficiency when querying current loans.

```
CREATE TABLE paymentHistory (
   pastLoanId NUMBER(9) PRIMARY KEY,
   fines NUMBER(6,2),
   FOREIGN KEY (pastLoanId)
      REFERENCES pastLoan(pastLoanId)
);
```

Triggers

To allow the database system to implement operational rules, maintain the logical integrity of data and ensure automatic data updating following certain data manipulations, it is necessary to take advantage of triggers. We included the following 5 representative triggers as follows.

Prevent unauthorized check-outs
When members attempt to check out resources, the database system should verify if they are permitted to do so. There are mainly 4 scenarios where members will be forbidden from checking out resources.

1. The borrower is suspended. According to operational rules of the library, suspended borrowers are barred from borrowing resources. This is done by checking the suspended status of the member when the system attempts to insert a new record in the currentLoan table.

2. The borrower is inactive. Inactive members are those who are no longer with the library but whose information is retained in the system for future reference or in case they return. They should not be allowed to check out resources unless they become active members again. This is done by checking the member type of the person before new records are inserted on the currentLoan table.

3. The borrower has used up his or her quota. Since members cannot borrow beyond their quota associated with their types of membership, the trigger will compare the number of outstanding resource copies a member has with his or her total quota and prevent any resources from being checked out when the quota is used up.

4. Resources are to be used in the library or are no longer under the possession of the library. Before inserting a new record in the currentLoan table, the trigger checks if the maximum loan period of that resource is 0 (resource can only be used on the premise) or –1 (resource not available) and, if so, prevents the insertion.

```
CREATE OR REPLACE TRIGGER noCheckout
BEFORE INSERT ON currentLoan
FOR EACH ROW
DECLARE
isSuspended NUMBER(1);
notActive VARCHAR2(40);
resourcePeriod NUMBER;
mQuota NUMBER;
usedQuota NUMBER;
BEGIN

SELECT suspendedStatus INTO isSuspended FROM member where cardNo=:NEW.cardNo;

IF isSuspended=1 THEN
raise_application_error(-1001,'Member is suspended. Check-out is forbidden.');
END IF;
```

```
SELECT mType INTO notActive FROM member where cardNo=:NEW.cardNo;

IF notActive='inactive' THEN
raise_application_error(-1002,'Member is inactive. Check-out is forbidden.');
END IF;

SELECT loanQuota INTO mQuota FROM member mb INNER JOIN memberType mt ON
mb.mType=mt.mType WHERE mb.cardNo=:NEW.cardNo;

SELECT COUNT(currentLoanID) INTO usedQuota FROM currentLoan WHERE
cardNo=:NEW.cardNo AND checkinDate is NULL;

IF usedQuota=mQuota THEN
raise_application_error(-1003,'Quota used up.');
END IF;

SELECT maxLoanPeriod INTO resourcePeriod FROM resources where
resourceId=:NEW.resourceId;
IF resourcePeriod in (0,-1) THEN
raise_application_error(-2001,'Resource is not available for check-out.');
END IF;

END;
```

Archive loan records and update member status upon check-in
As per our database design, the current loan table stores information about resources not checked in and those that are checked in but have unpaid fines. This trigger serves to address three scenarios upon resource check-in, i.e., when attempts are made to update the check-in date attribute in the current loan table.

1. A member checks in a resource and there is no fine. In this case, the related loan record will be removed from the current loan table and a new record with the same information plus the check-in date will be inserted in the past loan table.
2. A member checks in a resource and clears a fine. In addition to the archiving steps described above, the fine amount and associated loan ID will be inserted into the payment history table for record-keeping purposes.
    2.1 As a fine is paid in this scenario and thus may impact a member's suspended status, the trigger also checks to see if the person has just cleared a fine of at least 10 dollars AND does not have other fines greater than or equal to 10 dollars, in which case the person's status will be changed from suspended to unsuspended.
3. A member checks in a resource that has a fine but does not pay for the fine. In this scenario, the trigger allows the check-in date to be updated in the current loan table. No other data manipulations are made.

Note that we need to perform data manipulations on the table we read, mutating table errors may occur. To circumvent this, we utilized a compound trigger.

```
CREATE OR REPLACE TRIGGER archiveLoan
FOR UPDATE OF checkinDate ON currentLoan COMPOUND TRIGGER
archiveLoanId NUMBER;
isPaid NUMBER;
BigFinesCount NUMBER;
IsBigFines NUMBER(1);
archiveCardNo CHAR(10);

BEFORE EACH ROW IS
BEGIN
IF :OLD.FINES>=10 THEN
IsBigFines:=1;
ArchiveCardNo:=:OLD.cardNo;
        END IF;

IF :NEW.fines=0 THEN
isPaid:=1;
archiveLoanId:=:OLD.currentLoanID;
INSERT INTO pastLoan VALUES (:OLD.currentLoanID, :OLD.CardNo, :OLD.ResourceId,
:OLD.CopyID, :OLD.CheckoutDate, :NEW.CheckinDate);
IF :OLD.fines!=0 THEN
        INSERT INTO paymentHistory values (:OLD.currentLoanID,  :OLD.Fines);
        END IF;
ELSE isPaid:=0;
END IF;
END BEFORE EACH ROW;

AFTER STATEMENT IS
BEGIN

IF isPaid=1 THEN
        DELETE FROM currentLoan where currentLoanId=archiveLoanId;
END IF;

IF isBigFines=1 THEN
SELECT COUNT(CURRENTLOANID) INTO BigFinesCount FROM CURRENTLOAN
WHERE (FINES>=10 AND CARDNO= archiveCardNo);
/*After fine >=10 is paid, update suspended status if applicable*/
IF BigFinesCount=0 THEN
UPDATE MEMBER SET SUSPENDEDSTATUS=0 WHERE CARDNO= archiveCardNo;
END IF;
END IF;

END AFTER STATEMENT;
END archiveLoan;
```

Archive loan records and update member status upon payment of fines for resources checked in previously

Since we allow members to check in resources without paying fines as a way to encourage members to return resources, we need to devise a trigger to archive loan records and, if appropriate, update the member's current status.

When a fine for a resource copy returned previously is paid, the related loan record will be removed from the current loan table and a new record with the same information apart from fines will be inserted in the past loan table. The amount of the fine and the related loan ID will be stored in the fines table.

Again, since a payment is made, member status might need to be updated. As per the same logic described above, the trigger checks to see if the person has just cleared a fine of at least 10 dollars AND does not have other fines greater than or equal to 10 dollars. If so, the person will be changed from suspended to unsuspended.

Once again, we made use of a compound trigger to evade the mutating table issue.

```
CREATE OR REPLACE TRIGGER payFines
FOR UPDATE OF fines ON currentLoan COMPOUND TRIGGER
archiveLoanId NUMBER;
isPaid NUMBER;
BigFinesCount NUMBER;
IsBigFines NUMBER(1);
archiveCardNo CHAR(10);

BEFORE EACH ROW IS
BEGIN
IF :OLD.FINES>=10 THEN
IsBigFines:=1;
ArchiveCardNo:=:OLD.cardNo;
        END IF;
IF :OLD.checkinDate IS NOT NULL THEN
        archiveLoanId:=:OLD.currentLoanID;
        isPaid:=1;
        INSERT INTO pastLoan VALUES (:OLD.currentLoanID, :OLD.CardNo,
:OLD.ResourceId, :OLD.CopyID, :OLD.CheckoutDate, :OLD.CheckinDate);
        INSERT INTO paymentHistory values (:OLD.currentLoanID, :OLD.Fines);
ELSE isPaid:=0;
END IF;
END BEFORE EACH ROW;


AFTER STATEMENT IS
BEGIN
IF isPaid=1 THEN
        DELETE FROM currentLoan where currentLoanId=archiveLoanId;
END IF;
/*After fine >=10 is paid, update suspended status if applicable*/
IF isBigFines=1 THEN
SELECT COUNT(CURRENTLOANID) INTO BigFinesCount FROM CURRENTLOAN
WHERE (FINES>=10 AND CARDNO= archiveCardNo);
```

```
IF BigFinesCount=0 THEN
UPDATE MEMBER SET SUSPENDEDSTATUS=0 WHERE CARDNO= archiveCardNo;
END IF;
END IF;

END AFTER STATEMENT;

END payFines;
```

Update member status after daily fine accruement
According to library rules, once an outstanding resource goes beyond its check-out date by more than its maximum loan period allowed, a fine of 1 dollar will accrue for each day beyond that point, after which certain members can have fines that increase from below 10 dollars to at least 10 dollars, a situation that calls for updating of their member status. To that end, a trigger has been created to make possible updates.
As is coded below, a previously unsuspended member is updated to suspended when one of his fines has increased from less than 10 dollars to at least 10 dollars.

After fines auto-increment every day, update suspended status to True for those with fines changing from <10 to >=10

```
CREATE OR REPLACE TRIGGER updateSuspendedStatus
AFTER UPDATE OF fines ON currentLoan
FOR EACH ROW
WHEN (OLD.FINES<10 AND NEW.FINES>=10)
DECLARE currentStatus NUMBER(1);
BEGIN
   SELECT SUSPENDEDSTATUS INTO CURRENTSTATUS FROM MEMBER WHERE
MEMBER.CARDNO=:NEW.CARDNO;
   IF CURRENTSTATUS=0 THEN
      UPDATE MEMBER SET SUSPENDEDSTATUS=1 WHERE
MEMBER.CARDNO=:NEW.CARDNO;
   END IF;
END;
```

Data

In this section, different set of test data will be presented. These datasets are delicately designed to assess the overall integrity of the database. Extreme cases may be included in order to test the robustness of our SQL database comprehensively.

Test data for memberType:

The sample dataset presented here simulates the process of inserting new roles. Since there is no foreign key in this table. The most important constraint to test is the primary key. Therefore, an insert statement is deliberately designed to test if a repeated member type name will be rejected. And the test result shows if any repeating primary key value is being inserted to the table, the system will return an error message.

MTYPE	LOANQUOTA
student	5
staff	10
inactive	0
libstaff	12

```
INSERT INTO memberType VALUES ('staff',2);
/*Repeated member type, checking primary key constraint */
```

Test data for member:

Below set of data is the sample data for member table. In order to ensure the primary key is always in the format, we have included sample data where the memberId starts with a 0. It is confirmed that member ID could start with a 0 while the number of digits won’t change. Beneath the sample dataset, there are some statements presented in order to test other constraints.

CARDNO	MTYPE	FIRSTNAME	LASTNAME	SUSPENDEDSTATUS
1970478300	student	Kaylen	Burgess	0
1613553614	student	Joe	Farrington	0
0425661175	student	Reo	Bennett	0
4759330936	student	Miah	Mcdougall	0
7059502329	student	Alexis	Boyd	0
7484757945	student	Susan	Donovan	0
0402459128	student	Katelin	Jeffery	1
7499366446	inactive	Imaani	Ball	0
7076747074	student	Hope	Quintero	0
9371101492	staff	Christopher	Kim	0
4796850857	libstaff	Benas	Allan	0
5684760676	libstaff	Waqar	Lester	0
5045065061	libstaff	Tianna	Sinclair	0
0936481004	libstaff	Evie-Grace	Worthington	0
9177193960	libstaff	Lucas	Blackburn	0
6352392250	libstaff	Alicia	Hood	0

1965475927	inactive	Erik	Jordan	0
1945783451	student	Natahsha	Hakim	1
1945723752	staff	Dania	Willbald	0
1724738663	student	Golan	Kempton	1
1742566118	student	Oliva	Parkins	1
1613433613	student	Abby	Radovan	0
1735929420	student	Gordon	Slovski	0
1638542671	student	Lillian	Hamnet	0

```
INSERT INTO member VALUES ('1738683974','student','tim','Solow',0);
/* first letter of the first name is not capital*/

INSERT INTO member VALUES ('1738683974','student','Tim','Solow',3);
 /* suspendedStatus is neither 0 or 1 */

INSERT INTO member VALUES ('17386839743','student','Tim','Solow',0);
/* cardNo length is larger than 10  */

INSERT INTO member VALUES ('1613553614','student','Tim','Solow',0);
/* repeated cardNo checking primary key constraint */

INSERT INTO member VALUES ('17386839','PHDstudent','Tim','Solow',0);
/* using mtype that is not included in the memberType table, checking foreign key
constraint*/
```

Test data for emailAddress:

The dataset presented below is the sample data for the emailAddress table. Included within the data are some different realistic examples in order to test the integrity of the system. For example, we have included tuples with different length of phone number and different area code.
The constraints that need to be tested in this table are firstly, the card number must exist in the member table since this is a foreign key. Secondly, the phoneNumber attribute should not allow any letters to be entered. Detailed descriptions are provided along with the test statement below.

EMAIL	CARDNO
KaylenMagentaBurgess@gmail.com	1970478300
jfarrington88@yahoo.com	1613553614
jfarrington_uk@hotmail.com	1613553614
reo_1997_uk@hotmail.com	0425661175
mmcdougall68@gmail.com	4759330936
alexis_b77@hotmail.com	7059502329
susuand1999@gmail.com	7484757945
katelinjeffgb@hotmail.com	402459128

iball543@yahoo.com	7499366446
hopequintero@gmail.com	7076747074
christkim70@gmail.com	9371101492
ballen@yahoo.com	4796850857
wlester99@gmail.com	5684760676
tianas_uni@gmail.com	5045065061
ew1988929@hotmail.com	0936481004
lbburn@gmail.com	9177193960
aliciah_238@gmail.com	6352392250
dw@outlook.com	1945723752
n465@outlook.com	1945783451
abby452@yahoo.com	1613553614
gsolo@gmail.com	1735929420
DW@outlook.com	1945723752
N456@outlook.com	1945783451
bby452@yahoo.com	1613553614
GSolo@gmail.com	1735929420

INSERT INTO emailAddress VALUES ('TSolowgmail.com','1738683974');
/* email address does not contain '@' */

INSERT INTO emailAddress VALUES ('TSolowgmailcom','1738683974');
/* email address does not contain '.' */

INSERT INTO emailAddress VALUES ('GSolo@gmail.com','1738683974');
/* repeated email, checking primary constraint */

Test data for phoneNumber:

The dataset presented below is the sample data for the phoneNumber table. Included within the data are some different realistic examples in order to test the integrity of the system. For example, we have included tuples with different length of phone number and different area code.
The constraints that need to be tested in this table are firstly, the card number must exist in the member table since this is a foreign key. Secondly, the phoneNumber attribute should not allow any letters to be entered. Detailed descriptions are provided along with the test statement below.

COUNTRYCODE	PHONENO	CARDNO
1	1632960715	1970478300
44	1672960141	1613553614
44	1632960429	0425661175
44	1642966588	0425661175
44	1632960466	4759330936
86	1632960720	7059502329
88	164960656	7484757945
1	1632960639	0402459128
44	1632960472	7499366446

44	1682960945	7076747074
44	1632960377	9371101492
44	7879988203	4796850857
44	7069712564	5684760676
44	7837331585	5045065061
44	7866060219	0936481004
44	7873753738	9177193960
44	7747502644	6352392250
44	1468441741	1965475927
44	1246275268	1945723752

```
INSERT INTO phoneNumber VALUES (44,'01457691408','1738683974');
/* repeated cardNo, checking foreign key constraint */

INSERT INTO phoneNumber VALUES (44, 'abc457691408','1738683974');
/* letters instead of numbers in string value */
```

Test data for libRole

The sample dataset for libRole table included tuples that test the reference constraint on the attribute - role.

ROLE	READACCESS	WRITEACCESS
Library Assistant	0	0
Librarian	1	0
Library Cleaner	0	0
Senior Library Assistant	0	1
Finance Assistant	1	1
Library Manager	1	1
Data Librarian	1	1
Information Administrator	1	1

```
/*Constraint checking for libRole table */

INSERT INTO libRole VALUES ('Data Librarian', 1,1);
/* repeated role, checking primary key constraint */
```

Test data for libStaff

The sample dataset presented here simulates the process of inserting new library staff. Since both attributes in this table are foreign keys referring to other tables. The most important constraint to test is the reference constraints.

CARDNO	ROLE
4796850857	Librarian

6352392250	Librarian
5684760676	Senior Library Assistant
5045065061	Finance Assistant
0936481004	Library Manager
9177193960	Information Administrator

/*Constraint checking for libStaff table */

INSERT INTO libStaff VALUES ('4111111111', 'Librarian');
/* non-existent card number, checking foreign key constraint */

INSERT INTO libStaff VALUES ('1613553614', 'Headmaster');
/* non-existent role, checking foreign key constraint */

Test data for class:

CLASSCODE	**CLASSNAME**	**FLOORNUMBER**
MGAC	Management Accounting	1
ARIN	Artificial Intelligence	1
INFS	Information Systems	1
DASC	Data Science	2
PUBH	Public Health	1
MRKT	Marketing	2
NEUS	Neuroscience	0
RSKM	Risk Management	2
ENVS	Environmental Management	2
ZOOL	Zoology	0
HUAN	Human Anatomy	0

/* Constraint test for class table */

INSERT INTO class VALUES ('DASCI', 'Data Science', 2);
/* classCode is larger than ‘4’, checking length constraint */

INSERT INTO class VALUES ('ZOOL', 'Zoology', 3);
/* floorNumber not IN (0, 1, 2) */

Test data for shelf:

SHELFNUMBER	CLASSCODE
3	MRKT
3	NEUS

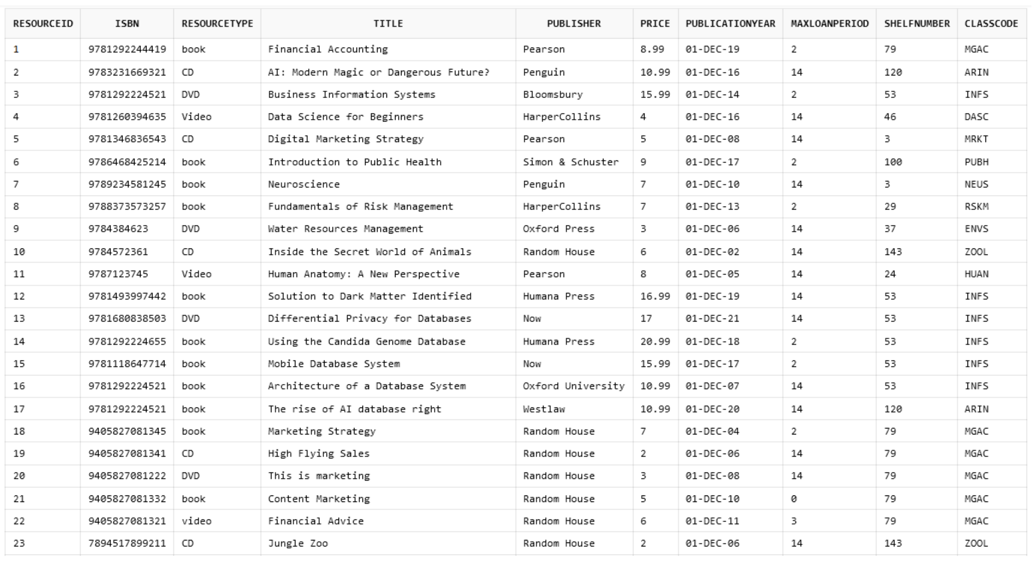
4	NEUS
24	HUAN
29	RSKM
37	ENVS
46	DASC
47	DASC
48	DASC
53	INFS
79	MGAC
100	PUBH
101	PUBH
120	ARIN
143	ZOOL

Constraint test for shelf

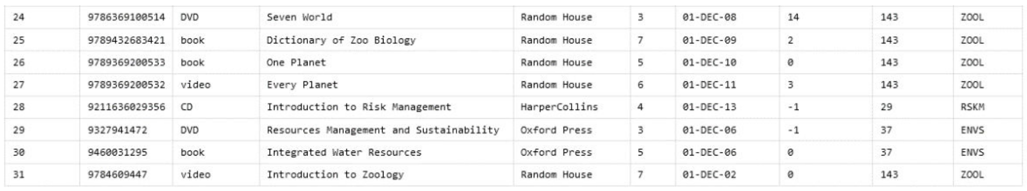
```
INSERT INTO shelf VALUES (79, 'MGAC');
/* repeated (shelfNumber, classCode), checking primary key constraint */
```

```
INSERT INTO shelf VALUES (79, 'MGC');
/* nonexistent classCode, checking foreign key constraint */
```

Test data for resources:
The dataset presented below is the sample data for the resources table. The constraints that need be tested in this table are firstly, the foreign key constraint; and secondly, the phoneNumber attribute should not allow any letters to be entered. Detailed descriptions are provided along with the test statement below.



RESOURCEID	ISBN	RESOURCETYPE	TITLE	PUBLISHER	PRICE	PUBLICATIONYEAR	MAXLOANPERIOD	SHELFNUMBER	CLASSCODE
1	9781292244419	book	Financial Accounting	Pearson	8.99	01-DEC-19	2	79	MGAC
2	9783231669321	CD	AI: Modern Magic or Dangerous Future?	Penguin	10.99	01-DEC-16	14	120	ARIN
3	9781292224521	DVD	Business Information Systems	Bloomsbury	15.99	01-DEC-14	2	53	INFS
4	9781260394635	Video	Data Science for Beginners	HarperCollins	4	01-DEC-16	14	46	DASC
5	9781346836543	CD	Digital Marketing Strategy	Pearson	5	01-DEC-08	14	3	MRKT
6	9786468425214	book	Introduction to Public Health	Simon & Schuster	9	01-DEC-17	2	100	PUBH
7	9789234581245	book	Neuroscience	Penguin	7	01-DEC-10	14	3	NEUS
8	9788373573257	book	Fundamentals of Risk Management	HarperCollins	7	01-DEC-13	2	29	RSKM
9	9784384623	DVD	Water Resources Management	Oxford Press	3	01-DEC-06	14	37	ENVS
10	9784572361	CD	Inside the Secret World of Animals	Random House	6	01-DEC-02	14	143	ZOOL
11	9787123745	Video	Human Anatomy: A New Perspective	Pearson	8	01-DEC-05	14	24	HUAN
12	9781493997442	book	Solution to Dark Matter Identified	Humana Press	16.99	01-DEC-19	14	53	INFS
13	9781680838503	DVD	Differential Privacy for Databases	Now	17	01-DEC-21	14	53	INFS
14	9781292224655	book	Using the Candida Genome Database	Humana Press	20.99	01-DEC-18	2	53	INFS
15	9781118647714	book	Mobile Database System	Now	15.99	01-DEC-17	2	53	INFS
16	9781292224521	book	Architecture of a Database System	Oxford University	10.99	01-DEC-07	14	53	INFS
17	9781292224521	book	The rise of AI database right	Westlaw	10.99	01-DEC-20	14	120	ARIN
18	9405827081345	book	Marketing Strategy	Random House	7	01-DEC-04	2	79	MGAC
19	9405827081341	CD	High Flying Sales	Random House	2	01-DEC-06	14	79	MGAC
20	9405827081222	DVD	This is marketing	Random House	3	01-DEC-08	14	79	MGAC
21	9405827081332	book	Content Marketing	Random House	5	01-DEC-10	0	79	MGAC
22	9405827081321	video	Financial Advice	Random House	6	01-DEC-11	3	79	MGAC
23	7894517899211	CD	Jungle Zoo	Random House	2	01-DEC-06	14	143	ZOOL



24	9786369100514	DVD	Seven World	Random House	3	01-DEC-08	14	143	ZOOL
25	9789432683421	book	Dictionary of Zoo Biology	Random House	7	01-DEC-09	2	143	ZOOL
26	9789369200533	book	One Planet	Random House	5	01-DEC-10	0	143	ZOOL
27	9789369200532	video	Every Planet	Random House	6	01-DEC-11	3	143	ZOOL
28	9211636029356	CD	Introduction to Risk Management	HarperCollins	4	01-DEC-13	-1	29	RSKM
29	9327941472	DVD	Resources Management and Sustainability	Oxford Press	3	01-DEC-06	-1	37	ENVS
30	9460031295	book	Integrated Water Resources	Oxford Press	5	01-DEC-06	0	37	ENVS
31	9784609447	video	Introduction to Zoology	Random House	7	01-DEC-02	0	143	ZOOL

INSERT INTO resources VALUES (32, '978943268341', 'book', 'Dictionary of Zoo', 'Random House', 7.00, To_DATE ('2009', 'YYYY'), 2, 143, 'ZOOL');
/* the number of character in ISBN is neither 10 or 13*/

INSERT INTO resources VALUES (33, '9789369200533', 'book', 'One Planet ', 'Random House', 5.00, To_DATE ('2010', 'YYYY'), 0, 111, 'ZOOL');
/* non-existent shelf number, checking foreign key constraint */

INSERT INTO resources VALUES (34, '9789369200532', 'video', 'Every Planet ', 'Random House', 6.00, To_DATE ('2011', 'YYYY'), 3, 143, 'WXYZ');
/* non-existent classCode, checking foreign key constraint */

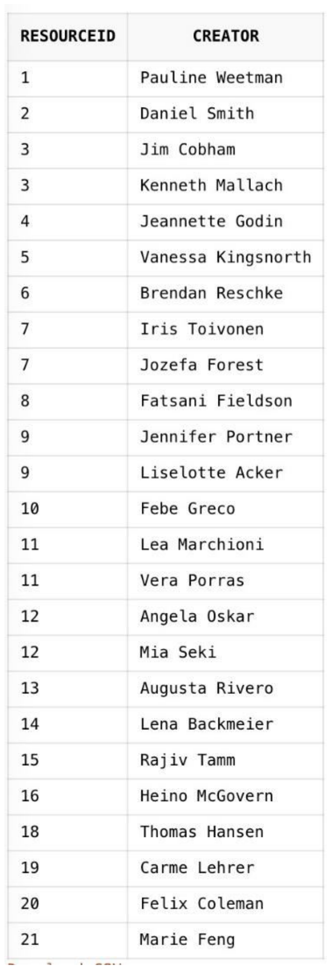
Test data for resourceCopy:
The constraint that needs to be tested in this table is the foreign key constraint on the resourceID.

RESOURCEID	COPYID
1	1
1	2
1	3
1	4
1	5
2	1
2	2
2	3
2	4
3	1
3	2
4	1
4	2
4	3
5	1
6	1
6	2
7	1
8	1
8	2
8	3
9	1
9	2
9	3
9	4
10	1
10	2
11	1
12	1
12	2
13	1
13	2
14	1
14	2
15	1
15	2
16	1
16	2
17	1

17	2
18	0
19	0
20	1
20	2
21	1

Test data for resourceCreator:

The sample data contains examples where a resource has multiple creators. Since a lot of resources are creator by multiple creators, it is important to verify that more than one creator can be added to a resource.
The constraint that has been tested in this table is the foreign key constraint on the resourceID.

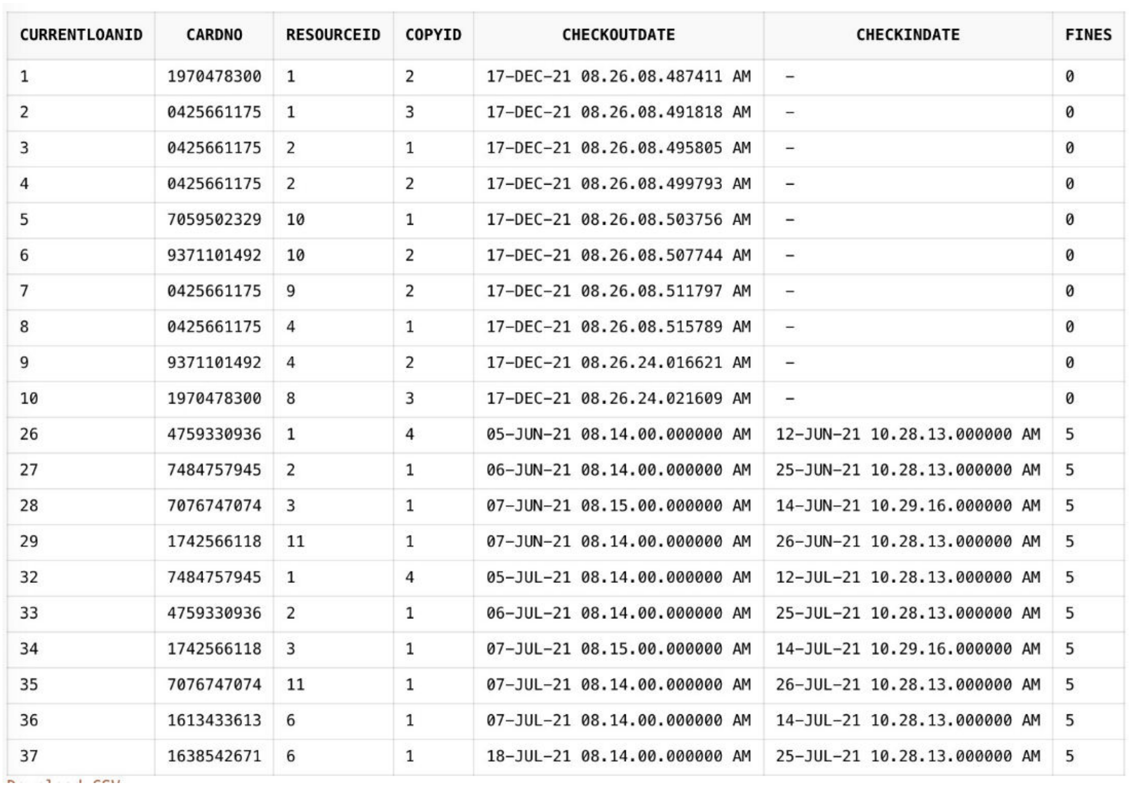


RESOURCEID	CREATOR
1	Pauline Weetman
2	Daniel Smith
3	Jim Cobham
3	Kenneth Mallach
4	Jeannette Godin
5	Vanessa Kingsnorth
6	Brendan Reschke
7	Iris Toivonen
7	Jozefa Forest
8	Fatsani Fieldson
9	Jennifer Portner
9	Liselotte Acker
10	Febe Greco
11	Lea Marchioni
11	Vera Porras
12	Angela Oskar
12	Mia Seki
13	Augusta Rivero
14	Lena Backmeier
15	Rajiv Tamm
16	Heino McGovern
18	Thomas Hansen
19	Carme Lehrer
20	Felix Coleman
21	Marie Feng

```
INSERT INTO resourceCreator VALUES (99, 'Steven Evans');
/* the resource id does not exist in the resources table, testing the foreign key constraint/
```

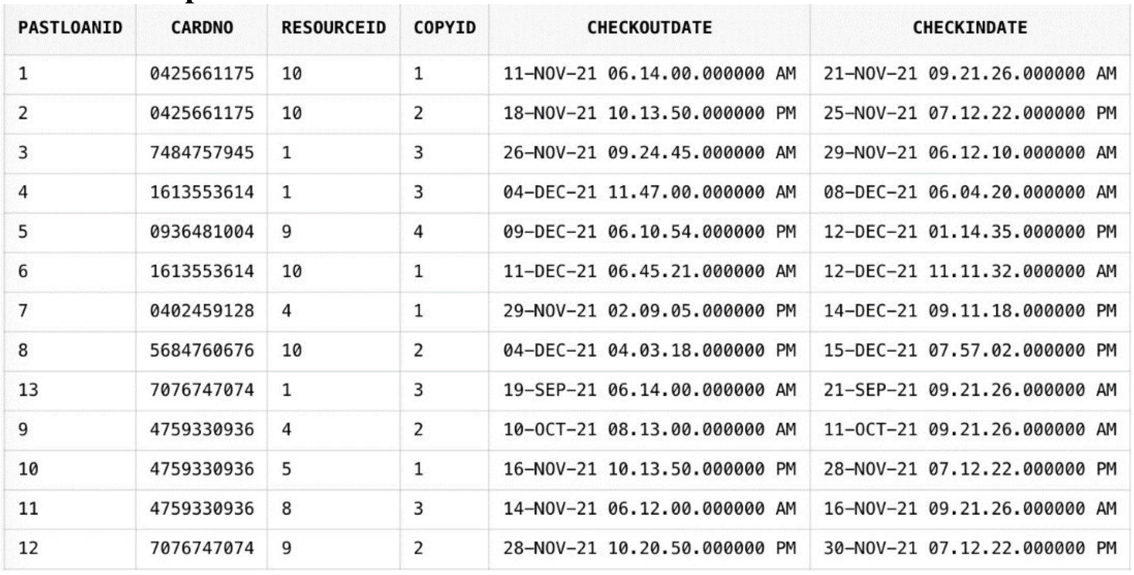
Test data for currentLoan:

The dataset presented below is the sample data for the currentLoan table.
The constraints that need be tested in this table are firstly, the foreign key constraint; and secondly, the phoneNumber attribute should not allow any letters to be entered. Detailed descriptions are provided along with the test statement below.



CURRENTLOANID	CARDNO	RESOURCEID	COPYID	CHECKOUTDATE	CHECKINDATE	FINES
1	1970478300	1	2	17-DEC-21 08.26.08.487411 AM	-	0
2	0425661175	1	3	17-DEC-21 08.26.08.491818 AM	-	0
3	0425661175	2	1	17-DEC-21 08.26.08.495805 AM	-	0
4	0425661175	2	2	17-DEC-21 08.26.08.499793 AM	-	0
5	7059502329	10	1	17-DEC-21 08.26.08.503756 AM	-	0
6	9371101492	10	2	17-DEC-21 08.26.08.507744 AM	-	0
7	0425661175	9	2	17-DEC-21 08.26.08.511797 AM	-	0
8	0425661175	4	1	17-DEC-21 08.26.08.515789 AM	-	0
9	9371101492	4	2	17-DEC-21 08.26.24.016621 AM	-	0
10	1970478300	8	3	17-DEC-21 08.26.24.021609 AM	-	0
26	4759330936	1	4	05-JUN-21 08.14.00.000000 AM	12-JUN-21 10.28.13.000000 AM	5
27	7484757945	2	1	06-JUN-21 08.14.00.000000 AM	25-JUN-21 10.28.13.000000 AM	5
28	7076747074	3	1	07-JUN-21 08.15.00.000000 AM	14-JUN-21 10.29.16.000000 AM	5
29	1742566118	11	1	07-JUN-21 08.14.00.000000 AM	26-JUN-21 10.28.13.000000 AM	5
32	7484757945	1	4	05-JUL-21 08.14.00.000000 AM	12-JUL-21 10.28.13.000000 AM	5
33	4759330936	2	1	06-JUL-21 08.14.00.000000 AM	25-JUL-21 10.28.13.000000 AM	5
34	1742566118	3	1	07-JUL-21 08.15.00.000000 AM	14-JUL-21 10.29.16.000000 AM	5
35	7076747074	11	1	07-JUL-21 08.14.00.000000 AM	26-JUL-21 10.28.13.000000 AM	5
36	1613433613	6	1	07-JUL-21 08.14.00.000000 AM	14-JUL-21 10.28.13.000000 AM	5
37	1638542671	6	1	18-JUL-21 08.14.00.000000 AM	25-JUL-21 10.28.13.000000 AM	5

Test data for pastLoan:



PASTLOANID	CARDNO	RESOURCEID	COPYID	CHECKOUTDATE	CHECKINDATE
1	0425661175	10	1	11-NOV-21 06.14.00.000000 AM	21-NOV-21 09.21.26.000000 AM
2	0425661175	10	2	18-NOV-21 10.13.50.000000 PM	25-NOV-21 07.12.22.000000 PM
3	7484757945	1	3	26-NOV-21 09.24.45.000000 AM	29-NOV-21 06.12.10.000000 AM
4	1613553614	1	3	04-DEC-21 11.47.00.000000 AM	08-DEC-21 06.04.20.000000 AM
5	0936481004	9	4	09-DEC-21 06.10.54.000000 PM	12-DEC-21 01.14.35.000000 PM
6	1613553614	10	1	11-DEC-21 06.45.21.000000 AM	12-DEC-21 11.11.32.000000 AM
7	0402459128	4	1	29-NOV-21 02.09.05.000000 PM	14-DEC-21 09.11.18.000000 PM
8	5684760676	10	2	04-DEC-21 04.03.18.000000 PM	15-DEC-21 07.57.02.000000 PM
13	7076747074	1	3	19-SEP-21 06.14.00.000000 AM	21-SEP-21 09.21.26.000000 AM
9	4759330936	4	2	10-OCT-21 08.13.00.000000 AM	11-OCT-21 09.21.26.000000 AM
10	4759330936	5	1	16-NOV-21 10.13.50.000000 PM	28-NOV-21 07.12.22.000000 PM
11	4759330936	8	3	14-NOV-21 06.12.00.000000 AM	16-NOV-21 09.21.26.000000 AM
12	7076747074	9	2	28-NOV-21 10.20.50.000000 PM	30-NOV-21 07.12.22.000000 PM

Test data for paymentHistory
In real life situation, this table should only contain data that is transferred from the currentLoan table by trigger. Therefore, there is no additional constraint needed to be tested. However, we are inserting manual data to the table to simulate a system that has been running for a while.

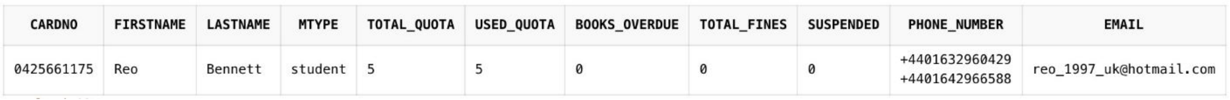
PASTLOANID	FINES
2	5
3	2
1	15
7	1
5	12

Views

Views serve the purpose of optimizing the database experience by displaying a subset of data that might be relevant to a particular user. Views can also be useful for security reasons by allowing users to obtain data without having to grant them access to the underlying tables in the database. Conclusively, three views have been created as shown below:

Personal statistics for members
This view provides a snapshot of a member's general information, including his or her personal details, contact information, membership type, used quota and total quota, as well as total fines and whether he or she is suspended. The following code is based on one member with a card number of 0425661175 as an example.

```
CREATE VIEW MemberProfile AS
/*Use WITH to give aliases to select results and enable referencing the results later in the
JOIN section*/
WITH MEMAIL AS
 (SELECT CARDNO, LISTAGG(EMAIL, CHR(10))  WITHIN GROUP (ORDER BY
EMAIL) AS EMAIL FROM EMAILADDRESS WHERE
EMAILADDRESS.CARDNO='0425661175' GROUP BY CARDNO),
MPHONE AS
 (SELECT CARDNO, LISTAGG('+'||COUNTRYCODE||PHONENO, CHR(10))  WITHIN
GROUP (ORDER BY COUNTRYCODE) AS PHONE_NUMBER FROM
PHONENUMBER WHERE CARDNO='0425661175' GROUP BY CARDNO),
MQUOTA AS
 (SELECT CARDNO, MT.MTYPE, LOANQUOTA AS TOTAL_QUOTA FROM
MEMBERTYPE MT INNER JOIN MEMBER MB ON MT.MTYPE=MB.MTYPE WHERE
CARDNO='0425661175'),
MUQUOTA AS
(SELECT CARDNO, COUNT(CARDNO)-1 AS USED_QUOTA FROM ((SELECT
CARDNO FROM CURRENTLOAN WHERE CARDNO='0425661175' AND
CHECKINDATE IS NULL) UNION ALL (SELECT CARDNO FROM MEMBER WHERE
CARDNO='0425661175')) GROUP BY CARDNO),
MOVERDUE AS
(SELECT CARDNO, COUNT(CARDNO)-1 AS BOOKS_OVERDUE FROM ((SELECT
CARDNO, RESOURCEID FROM CURRENTLOAN WHERE CARDNO='0425661175'
AND FINES>=10) UNION ALL (SELECT CARDNO, CARDNO FROM MEMBER
WHERE CARDNO='0425661175')) GROUP BY CARDNO),
MTFINES AS
(SELECT CARDNO, SUM(FINES) AS TOTAL_FINES FROM ((SELECT CARDNO,
FINES FROM CURRENTLOAN WHERE CARDNO='0425661175') UNION ALL
(SELECT CARDNO, LENGTH(CARDNO)-LENGTH(CARDNO) FROM MEMBER
WHERE CARDNO='0425661175')) GROUP BY CARDNO)
SELECT MEMBER.CARDNO,FIRSTNAME,LASTNAME,
MEMBER.MTYPE,TOTAL_QUOTA, USED_QUOTA,
BOOKS_OVERDUE,TOTAL_FINES,SUSPENDEDSTATUS AS SUSPENDED,
PHONE_NUMBER, EMAIL
FROM MEMBER INNER JOIN MEMAIL ON MEMBER.CARDNO =
MEMAIL.CARDNO INNER JOIN MPHONE ON MEMAIL.CARDNO=
MPHONE.CARDNO INNER JOIN MQUOTA ON
MPHONE.CARDNO=MQUOTA.CARDNO INNER JOIN MUQUOTA ON
MQUOTA.CARDNO=MUQUOTA.CARDNO INNER JOIN MOVERDUE ON
MUQUOTA.CARDNO= MOVERDUE.CARDNO INNER JOIN MTFINES ON
MOVERDUE.CARDNO=MTFINES.CARDNO;
```

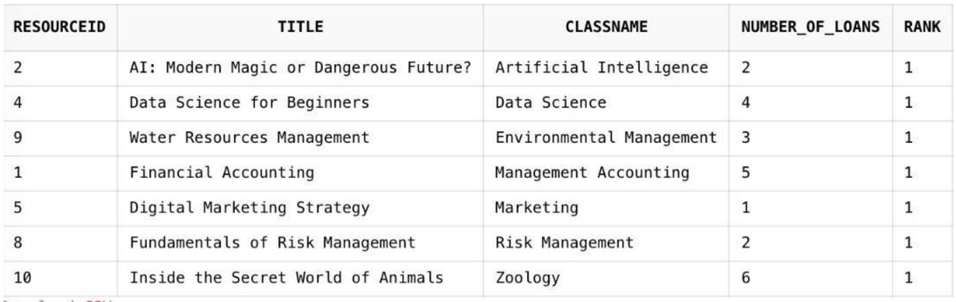


CARDNO	FIRSTNAME	LASTNAME	MTYPE	TOTAL_QUOTA	USED_QUOTA	BOOKS_OVERDUE	TOTAL_FINES	SUSPENDED	PHONE_NUMBER	EMAIL
0425661175	Reo	Bennett	student	5	5	0	0	0	+4401632960429 +4401642966588	reo_1997_uk@hotmail.com

Popular books by subject area

This view is useful to members of the library as it allows them to discover what books are trending in different subject areas. They can discover popular books of a genre they might be interested in. This view displays the top 5 books that have been checked out in the last 3 months, grouped by class name. By combining both current loans and past loans that have been checked out in the past 3 months, the top 5 books of each class are displayed, ordered by the number of times the per class a book has been checked out and ranked from 1 to 5.

```
CREATE VIEW trendingResourcesByGenre AS
SELECT * FROM
(SELECT totalLoans.resourceId, resources.title, class.className, COUNT(*) as
number_of_loans, row_number()OVER (
        PARTITION BY class.className
        ORDER BY count(*) DESC
    )rank
FROM
(SELECT resourceId from pastLoan WHERE pastLoan.checkoutDate >
add_months(CURRENT_TIMESTAMP, -3)
UNION ALL
SELECT resourceId from currentLoan WHERE currentLoan.checkoutDate >
add_months(CURRENT_TIMESTAMP, -3))totalLoans
JOIN resources ON resources.resourceId = totalLoans.resourceId
JOIN class ON resources.classCode = class.classCode
GROUP BY totalLoans.resourceId, resources.title, class.className
ORDER BY class.className, number_of_loans DESC)
WHERE rank <= 5;
```

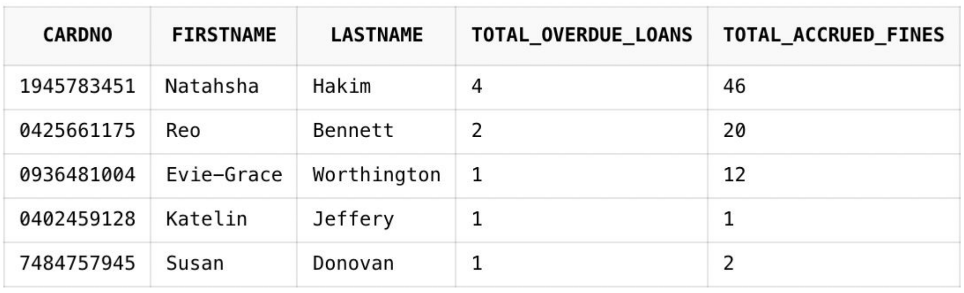


RESOURCEID	TITLE	CLASSNAME	NUMBER_OF_LOANS	RANK
2	AI: Modern Magic or Dangerous Future?	Artificial Intelligence	2	1
4	Data Science for Beginners	Data Science	4	1
9	Water Resources Management	Environmental Management	3	1
1	Financial Accounting	Management Accounting	5	1
5	Digital Marketing Strategy	Marketing	1	1
8	Fundamentals of Risk Management	Risk Management	2	1
10	Inside the Secret World of Animals	Zoology	6	1

List of persistent offenders sorted by the number of payments made by each member

This view allows the library staff to identify the persistent offenders (i.e. members that have returned the resources late multiple times). This is useful information for both daily library operations and longer-term library planning. Because the library can then remind the persistent offenders to return the items on time or even suspend any serious offenders. In the code provided below, it is shown that inactive members are excluded. Also, the number of times the member had returned a resource late and the total amount of fines he/she made are shown in the query result. The result is sorted by the number of payments made by each member. It gives the library staff convenience in identifying the top offenders.

```
CREATE VIEW listPersistentOffenders AS
SELECT member.cardNo, member.firstName, member.lastName, COUNT(*) as total_overdue_loans,
SUM(fines) AS total_accrued_fines
FROM paymentHistory
JOIN pastLoan ON pastLoan.pastLoanId = paymentHistory.pastLoanId
JOIN member ON pastLoan.cardNo = member.cardNo
WHERE member.mType != 'inactive'
GROUP BY member.cardNo, member.firstName, member.lastName
UNION ALL
SELECT member.cardNo, member.firstName, member.lastName, COUNT(*) as total_overdue_loans,
SUM(fines) AS total_accrued_fines
FROM currentLoan
JOIN member ON member.cardNo = currentLoan.cardNo
WHERE currentLoan.fines > 0
GROUP BY member.cardNo, member.firstName, member.lastName
ORDER BY total_overdue_loans DESC
```

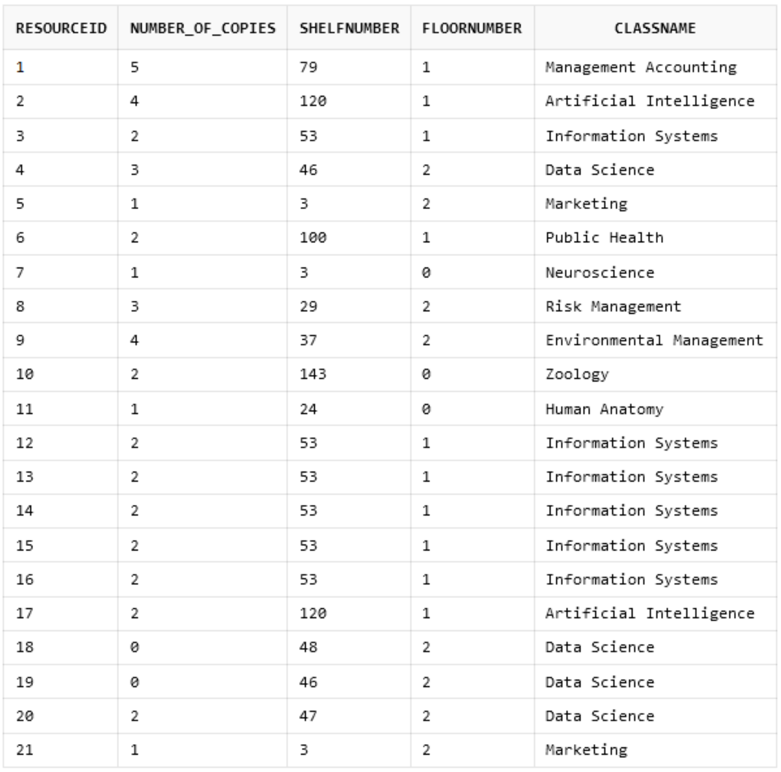


CARDNO	FIRSTNAME	LASTNAME	TOTAL_OVERDUE_LOANS	TOTAL_ACCRUED_FINES
1945783451	Natahsha	Hakim	4	46
0425661175	Reo	Bennett	2	20
0936481004	Evie-Grace	Worthington	1	12
0402459128	Katelin	Jeffery	1	1
7484757945	Susan	Donovan	1	2

Queries

Display all resources and their respective class number, number of copies, and location.
This query gives an overview of all resources in the library and their respective class number, number of copies, and location. It is a useful query for situations such as when the library staff want to do a thorough library check. It can also serve as a catalogue of the library resources. As the code below shows, the query is done by selecting the necessary columns from the Resource table and the Resource Copy table.

```
SELECT resourceCopy.resourceId, MAX(resourceCopy.copyId) AS number_of_copies,
        resources.shelfNumber, class.floornumber, class.className
FROM resourceCopy
JOIN resources ON resources.resourceId = resourceCopy.resourceId
JOIN class ON resources.classCode = class.classCode
GROUP BY resourceCopy.resourceId, resources.shelfNumber, class.floornumber,
        class.className
ORDER BY resourceId;
```

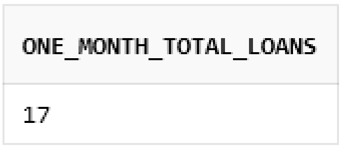


RESOURCEID	NUMBER_OF_COPIES	SHELFNUMBER	FLOORNUMBER	CLASSNAME
1	5	79	1	Management Accounting
2	4	120	1	Artificial Intelligence
3	2	53	1	Information Systems
4	3	46	2	Data Science
5	1	3	2	Marketing
6	2	100	1	Public Health
7	1	3	0	Neuroscience
8	3	29	2	Risk Management
9	4	37	2	Environmental Management
10	2	143	0	Zoology
11	1	24	0	Human Anatomy
12	2	53	1	Information Systems
13	2	53	1	Information Systems
14	2	53	1	Information Systems
15	2	53	1	Information Systems
16	2	53	1	Information Systems
17	2	120	1	Artificial Intelligence
18	0	48	2	Data Science
19	0	46	2	Data Science
20	2	47	2	Data Science
21	1	3	2	Marketing

Number of total loans checked out in the past month(s)
This query allows users to check the number of total loans checked out in the previous month(s). The code below shows an example of counting the number of loans checked out in the past month. This is a common feature for the library staff to get a brief idea of how many loans have been made in the past month(s). Notice that in the code there is no table required after the FROM clause, hence FROM dual is used for this case.

```
SELECT
(SELECT COUNT(*) FROM currentLoan WHERE checkoutDate >
        add_months(current_timestamp,-1))+
(SELECT COUNT(*) from pastLoan WHERE checkoutDate >
        add_months(current_timestamp,-1))
```

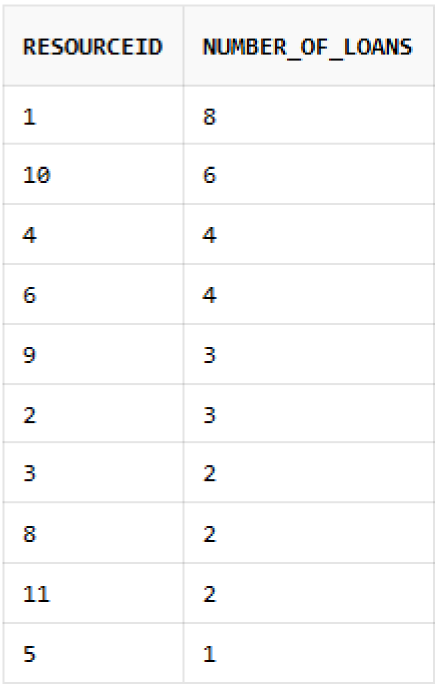
```
AS one_month_total_loans FROM dual;
```



ONE_MONTH_TOTAL_LOANS
17

Listing all resource items according to the popularity in the past year(s)
This is a query that shows the number of times each resource has been borrowed in the past year(s). The table generated is concise and easy to ready. There are only two columns where the first column shows the id of all the resources that have been borrowed at least once; and the second column shows the number of times that resources have been borrowed. And the result is sorted by the popularity of the resources. The query can be modified to show the results of a different length of time (e.g. 24 or 36 months). This information is helpful to library management in making decisions on what resources they should buy more copies for.

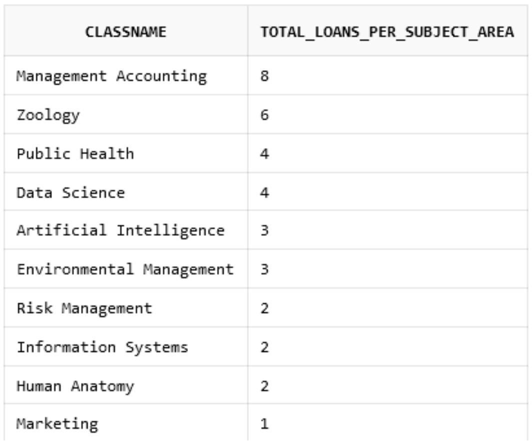
```
SELECT resourceId, COUNT(*) AS number_of_loans
FROM
        (SELECT resourceId from pastLoan WHERE checkoutDate >
                add_months(CURRENT_TIMESTAMP, -12)
        UNION ALL
        SELECT resourceId from currentLoan WHERE checkoutDate >
                add_months(CURRENT_TIMESTAMP, -12))
GROUP BY resourceId
ORDER BY number_of_loans DESC;
```



RESOURCEID	NUMBER_OF_LOANS
1	8
10	6
4	4
6	4
9	3
2	3
3	2
8	2
11	2
5	1

Ranking of most popular subject areas
This is a query that allows the library to check the popularity of all the classes. The code below shows that this query counts the number of times each resource has been borrowed, both previously and currently. It gives the library staff an idea of which subject areas are popular and helps them decide which classes need more resources.

```
SELECT class.className, COUNT(totalLoans.resourceId) AS
            total_loans_per_subject_area FROM
            (SELECT resourceId from pastLoan
            UNION ALL
            SELECT resourceId from currentLoan)totalLoans
JOIN resources ON resources.resourceId = totalLoans.resourceId
JOIN class ON resources.classCode = class.classCode
GROUP BY class.className
ORDER BY total_loans_per_subject_area DESC
```

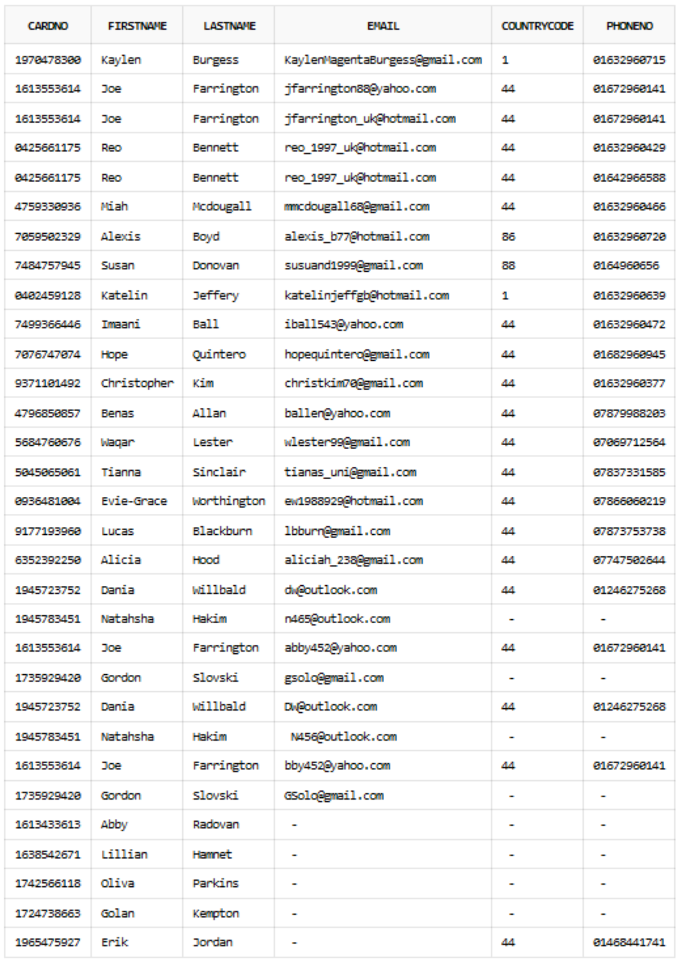


CLASSNAME	TOTAL_LOANS_PER_SUBJECT_AREA
Management Accounting	8
Zoology	6
Public Health	4
Data Science	4
Artificial Intelligence	3
Environmental Management	3
Risk Management	2
Information Systems	2
Human Anatomy	2
Marketing	1

List the contact details of all members

This is a query that presents the phone number and email address of all members. The code for this query is straightforward as it can be done by simply joining the Member table with Email Address and Phone Number table. It gives the library staff a quick way to look for the contact information of any members.

```
SELECT m.cardNo, m.firstName, m.lastName, emailAddress.email, phoneNumber.countryCode,
phoneNumber.phoneNo
FROM member m
LEFT JOIN emailAddress ON m.cardNo = emailAddress.cardNo
LEFT JOIN phoneNumber ON m.cardNo = phoneNumber.cardNo;
```

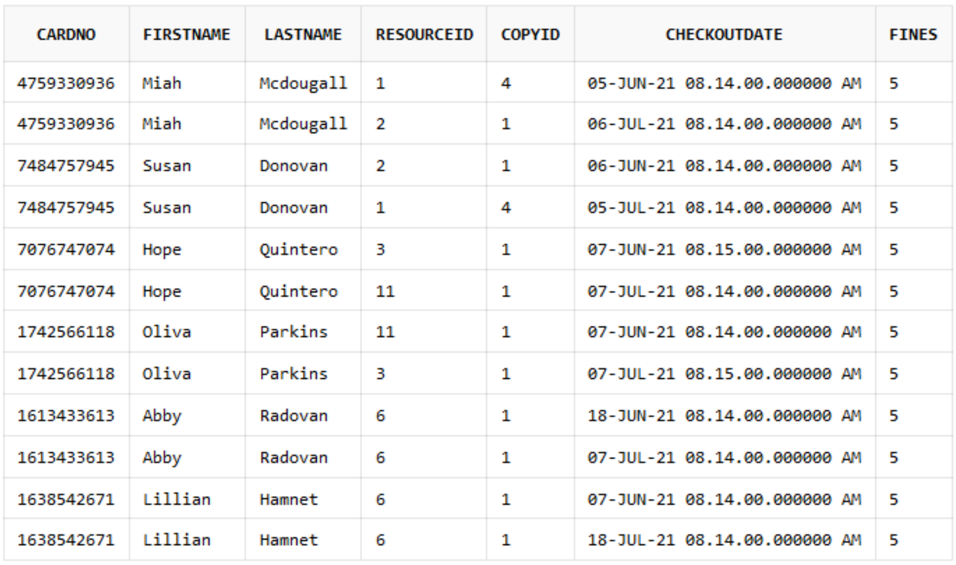


CARDNO	FIRSTNAME	LASTNAME	EMAIL	COUNTRYCODE	PHONENO
1970478300	Kaylen	Burgess	KaylenMagentaBurgess@gmail.com	1	01632960715
1613553614	Joe	Farrington	jfarrington88@yahoo.com	44	01672960141
1613553614	Joe	Farrington	jfarrington_uk@hotmail.com	44	01672960141
0425661175	Reo	Bennett	reo_1997_uk@hotmail.com	44	01632960429
0425661175	Reo	Bennett	reo_1997_uk@hotmail.com	44	01642966588
4759330936	Miah	Mcdougall	mmcdougall68@gmail.com	44	01632960466
7059502329	Alexis	Boyd	alexis_b77@hotmail.com	86	01632960720
7484757945	Susan	Donovan	susuand1999@gmail.com	88	0164960656
0402459128	Katelin	Jeffery	katelinjeffgb@hotmail.com	1	01632960639
7499366446	Imaani	Ball	iball543@yahoo.com	44	01632960472
7076747074	Hope	Quintero	hopequintero@gmail.com	44	01682960945
9371101492	Christopher	Kim	christkim70@gmail.com	44	01632960377
4796850857	Benas	Allan	ballen@yahoo.com	44	07879988203
5684760676	Waqar	Lester	wlester99@gmail.com	44	07069712564
5045065061	Tianna	Sinclair	tianas_uni@gmail.com	44	07837331585
0936481004	Evie-Grace	Worthington	ew1988929@hotmail.com	44	07866060219
9177193960	Lucas	Blackburn	lbburn@gmail.com	44	07873753738
6352392250	Alicia	Hood	aliciah_238@gmail.com	44	07747502644
1945723752	Dania	Willbald	dw@outlook.com	44	01246275268
1945783451	Natahsha	Hakim	n465@outlook.com	-	-
1613553614	Joe	Farrington	abby452@yahoo.com	44	01672960141
1735929420	Gordon	Slovski	gsolo@gmail.com	-	-
1945723752	Dania	Willbald	Dw@outlook.com	44	01246275268
1945783451	Natahsha	Hakim	N456@outlook.com	-	-
1613553614	Joe	Farrington	bby452@yahoo.com	44	01672960141
1735929420	Gordon	Slovski	GSolo@gmail.com	-	-
1613433613	Abby	Radovan	-	-	-
1638542671	Lillian	Hamnet	-	-	-
1742566118	Oliva	Parkins	-	-	-
1724738663	Golan	Kempton	-	-	-
1965475927	Erik	Jordan	-	44	01468441741

Find the members with outstanding fines

This query presents the details of all the members who have outstanding fines and the corresponding overdue resource items. This is a helpful function for library staff to check which member is currently owing the library fines. This information allows the staff to perform follow-up actions such as sending reminders to these members. The code shows that it works by simply finding the member with fines in the Current Loan table.

```
SELECT member.cardNo, member.firstName, member.lastName, currentLoan.resourceId,
        currentLoan.copyId, currentLoan.checkoutDate, currentLoan.fines
FROM currentLoan
JOIN member ON  currentLoan.cardNo =  member.cardNo
WHERE (currentLoan.fines > 0)
```

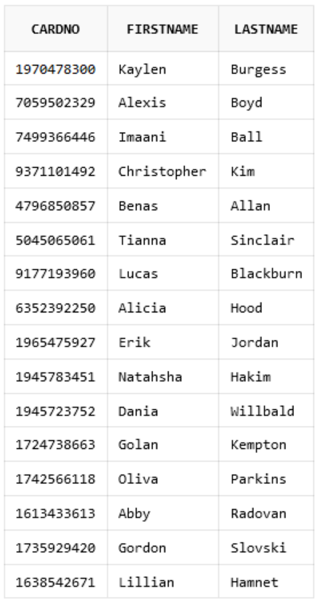


CARDNO	FIRSTNAME	LASTNAME	RESOURCEID	COPYID	CHECKOUTDATE	FINES
4759330936	Miah	Mcdougall	1	4	05-JUN-21 08.14.00.000000 AM	5
4759330936	Miah	Mcdougall	2	1	06-JUL-21 08.14.00.000000 AM	5
7484757945	Susan	Donovan	2	1	06-JUN-21 08.14.00.000000 AM	5
7484757945	Susan	Donovan	1	4	05-JUL-21 08.14.00.000000 AM	5
7076747074	Hope	Quintero	3	1	07-JUN-21 08.15.00.000000 AM	5
7076747074	Hope	Quintero	11	1	07-JUL-21 08.14.00.000000 AM	5
1742566118	Oliva	Parkins	11	1	07-JUN-21 08.14.00.000000 AM	5
1742566118	Oliva	Parkins	3	1	07-JUL-21 08.15.00.000000 AM	5
1613433613	Abby	Radovan	6	1	18-JUN-21 08.14.00.000000 AM	5
1613433613	Abby	Radovan	6	1	07-JUL-21 08.14.00.000000 AM	5
1638542671	Lillian	Hamnet	6	1	07-JUN-21 08.14.00.000000 AM	5
1638542671	Lillian	Hamnet	6	1	18-JUL-21 08.14.00.000000 AM	5

List of members who haven't borrowed a book

This query displays a list of members who have never borrowed any resources. It allows the library staff or management to know who idle members are. This type of information can provide valuable statistics when it is combined with other data stored in a different university database (i.e. external to this library system). For example, by knowing the name of the idle students, the user can calculate the statistics about students from which subjects access the library resources the least.

```
SELECT member.cardNo, member.firstName, member.lastName
FROM
        (SELECT member.cardNo FROM member
        MINUS
        SELECT pastLoan.cardNo FROM pastLoan)idleMember
JOIN member ON member.cardNo = idleMember.cardNo
```

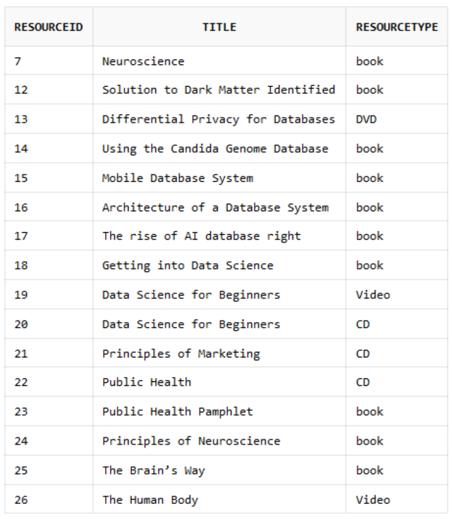


CARDNO	FIRSTNAME	LASTNAME
1970478300	Kaylen	Burgess
7059502329	Alexis	Boyd
7499366446	Imaani	Ball
9371101492	Christopher	Kim
4796850857	Benas	Allan
5045065061	Tianna	Sinclair
9177193960	Lucas	Blackburn
6352392250	Alicia	Hood
1965475927	Erik	Jordan
1945783451	Natahsha	Hakim
1945723752	Dania	Willbald
1724738663	Golan	Kempton
1742566118	Oliva	Parkins
1613433613	Abby	Radovan
1735929420	Gordon	Slovski
1638542671	Lillian	Hamnet

Find resources that have not been loaned for a certain period

The query presents the list of resources that have not been loaned for a certain period. This is a useful query for library manager to identify the resources that are not popular. With this function, the library manager can make decision on which resources may be discarded. The code below is an example of query selecting resources have not been loaned in the past three years.

```
SELECT resources.resourceId, resources.title, resources.resourceType
FROM
        (SELECT resourceId
        FROM resources
        MINUS
        (SELECT  resourceId
        FROM pastLoan
        WHERE pastLoan.checkoutDate > add_months(CURRENT_TIMESTAMP, -12*3)
        UNION
        SELECT resourceId
        FROM currentLoan))unusedResources
JOIN resources ON unusedResources.resourceId = resources.resourceId;
```

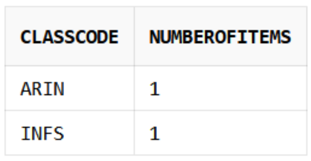


RESOURCEID	TITLE	RESOURCETYPE
7	Neuroscience	book
12	Solution to Dark Matter Identified	book
13	Differential Privacy for Databases	DVD
14	Using the Candida Genome Database	book
15	Mobile Database System	book
16	Architecture of a Database System	book
17	The rise of AI database right	book
18	Getting into Data Science	book
19	Data Science for Beginners	Video
20	Data Science for Beginners	CD
21	Principles of Marketing	CD
22	Public Health	CD
23	Public Health Pamphlet	book
24	Principles of Neuroscience	book
25	The Brain's Way	book
26	The Human Body	Video

Count of resources per classes

This query presents an example of finding the number of items per class based on the publication year and the floor number. This is useful when the library is doing a regular stock check. The variables of this query are the floor number and the publication year. In the code below, we are counting the number of resource items that is located on the first floor and was published between 2010 and 2017. The result is then presented by class codes.

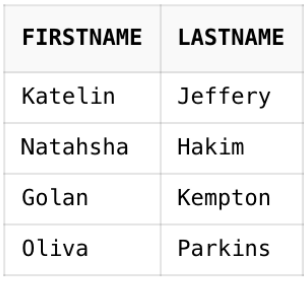
```
SELECT class2.classCode, COUNT (class2.classCode) as NumberOfItems
FROM resources
JOIN
   (SELECT classCode FROM class WHERE floorNumber = 1)class2
ON resources.classCode = class2.classCode
WHERE (resources.publicationYear >To_DATE ('2010', 'YYYY'))
   AND (resources.publicationYear <To_DATE ('2017', 'YYYY'))
GROUP BY class2.classCode
```



CLASSCODE	NUMBEROFITEMS
ARIN	1
INFS	1

List of people currently suspended
This query provides a list of members who are currently suspended.

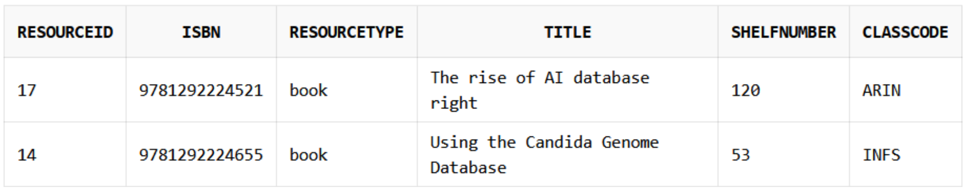
```
SELECT member.firstName, member.lastName
FROM member
WHERE suspendedStatus = 1
```



FIRSTNAME	LASTNAME
Katelin	Jeffery
Natahsha	Hakim
Golan	Kempton
Oliva	Parkins

Search a particular type of resources using keywords
This query allows user to search for the resources of a particular type by using keywords. As an additional feature, publication year can be specified for more refined search. It should be one of the most popular functions that the students will use. We are allowing user to search for items published after a specific publication year because students normally look for more updated materials. In the code below, we are searching for books which were published in or after 2017 and has the word ‘database’ in the title.

```
SELECT resourceid,ISBN, resourceType, title, shelfnumber, classcode
FROM resources
WHERE lower(title) like '%database%'
AND publicationYear> To_DATE ('2017', 'YYYY')
AND resourceType = 'book'
ORDER BY ISBN;
```



RESOURCEID	ISBN	RESOURCETYPE	TITLE	SHELFNUMBER	CLASSCODE
17	9781292224521	book	The rise of AI database right	120	ARIN
14	9781292224655	book	Using the Candida Genome Database	53	INFS

Find the location of resource of a subject area
This query displays the shelf and floor number of a class by using class name. It is a simple query that allows user to know the location of a particular class. Therefore, it will be another most popular function among library users. The code below shows an example of finding the location of the class named Information Systems.

```
SELECT class3.floorNumber, shelf.shelfNumber
FROM shelf
JOIN
   (SELECT classCode, floorNumber
   FROM class
   WHERE className= 'Information Systems')class3
ON shelf.classCode = class3.classCode;
```

FLOORNUMBER	SHELFNUMBER
1	53

Security

External threats to organizational database, personal and private data of the organisation, are historically the main concerns when it comes to data security. Insider threats, however, could be more difficult to prevent for a variety of reasons, one of which is that insiders may threaten the organization's data security unintentionally.

Considering personal and private data is at a very high risk of being compromised in terms of confidentiality, integrity, and availability, the UK government have put in place the Data Protection Act 2018 with which all organizations in the UK have to comply with. With 'Integrity and confidentiality' being one of the main principles, organizations are required to enforce a strict data security policy that must be proportionate to the type of data they possess.

In our library database system design, it does not store the most sensitive data of the users such as credit card details, hence theoretically the level of external threats should not be severe. However, the system records diverse types of users' personal details including their email, phone number and member id, a holistic policy to data security is still needed to minimise the risk of exposing to internal threats, whether intentional or unintentional.

A database security policy generally comprises three aspects – Authentication, Authorization, and Encryption (Sharma et al., 2017). It can protect the data to the greatest extent only when both the database administrators (DBA) and system user do their parts well. The basis of any security approach is authentication, which refer to the process of verifying the identity of a person attempting to access the system. The most authentication method is passwords, and this is the security policy component to which a system user has the most contribution. A system user must avoid using passwords that are easy to guess (e.g. a word from the dictionary). At the same time, is also a DBA's job to verify the use of strong password and implement account lockout after failed access attempts (Ben-Natan, 2009). For example, the use of attribute 'FAILED_LOGIN_ATTEMPTS' on Oracle allow the DBA to set the consecutive failed login attempts to the user account before the account is locked.

The second layer – authorization – refers to the controlling of access rights and privileges of different users to data. In this layer, the system users can give only very limited contribution because it is mainly DBA's duty to implement an appropriate access control. In the Oracle environment, the first step a DBA should do is to identify all the user roles for the system, then privileges can be assigned to each role according to the nature of the role. A rule of thumb is to prevent granting non-DBA user any system privileges such as UPDATE ANY TABLE. Applying to our library system design, only the administrator can have the SYS schema because it gives him/her explicit object privileges. In Oracle, for instance, the predefined role - EXP_FULL_DATABASE – should be possessed by only the administrator since this gives the privileges to perform full and incremental database export.

Aside from the privileges assigned to the predefined roles, DBA can even grant additional privileges to individual user to customise a unique set of access rights. A librarian, for example, should be able to enter and modify, but not delete, data in the system, because the right to delete a record should only be given to library manager. Therefore, system privilege, such as 'DELETE ANY TABLE', that allows any kind of deletion of data should only be granted to library manager's account.

Aside from defining the role and the privileges it has, a row-level security approach can also be used to manage the system access in an even more precise way. This is also known as the Fine-Grained Access Control in Oracle environment. It allows DBAs to add additional predicates to every SQL statement issued (Ben-Natan, 2009). In our library system, the students and library staffs should have different authority to data. The user access rights can be defined by using Oracle's Virtual Private Network, where DBA can add predicates to restrict users' query by simply creating a SQL function. For example, if the DBA wants to restrict students accessing the library database to see only their own accounts, he/she can create security policy functions attached to the 'Member', 'CurrentLoan', 'PastLoan' table, so that when a student (e.g. a student with card number as 1613553614) is accessing the tables using a SELECT query, a predicate "WHERE cardNo = 1613553614" will be automatically added to the query.

The third layer of database security policy - 'Encryption' – helps an organization in complying the 'confidentiality' principle of the UK Data Protection Act 2018. Encryption is the process of converting plain data into cypher text that can only be read by the sender and receiver. The purpose of encryption is to secure stored information and information transmission (Kohli, 2016). For our library system, since it is only expected to be used by people in the university, the exposure to the security threats during information transmission should be minimal. Therefore, more effort should be put on encrypting data-at-rest. In normal database system design, a DBA usually has unlimited access to any data, so in theory, a DBA can even modify the audit trail after he/she had done anything inappropriate with the data. It is, therefore, crucial to encrypt sensitive data so that the information leakage can be minimised, both internally and externally.

Even though encryption can hugely enhance the data security, attention needs to be paid to the selection of data to be encryption since any unnecessary encryption could severely impeded the system performance. Only the important or sensitive data should be encryption. Columns that are used as keys or indexes must not be encrypted as this will force table scans with decrypt functions. For example, in our library system, if we were to encrypt the personal data such as first name in the Member table, it is important to leave the primary key 'Card Number' unencrypted.

Apart from the three aspects of data security discussed above, other measures should be taken into consideration during system designing phrase in order to ensure compliance with the UK Data Protection Act 2018. As the Act states, organizations should minimise the data they collect and ensure any personal data collected will be stored no longer than necessary. It is why when designing our library database system, we have made sure only necessary data is stored. For example, information about members' email and phone number are stored because they are necessary for communication. Nevertheless, details such as students' family details or emergency contact details should not be stored in a library system as they should be kept in a separate university database system.

Bibliography

Ben-Natan, R., 2009. Implementing database security and auditing. Burlington, MA: Elsevier Digital Press. Data Protection Act 2018, c.12. Available at: https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted (Accessed: 15 December 2021).

Kohli, G., 2016. E-Commerce: transaction security issues and challenges. Clear International Journal of Research in Commerce & Management, 7(2), pp.91-93.

Sharma, P., Kaushik, P., Agarwal, P., Jain, P., Agarwal, S. and Dixit, K., 2017. Issues and challenges of data security in a cloud computing environment. 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON).