Library Database Management System

ECS740P – Database Systems

Coursework 2

Content

Introduction

In this report, we will use Oracle Live SQL to implement the database that we previously designed. This will be carried out by producing test data, representing the data with tables and views, inserting data into our tables and creating queries as a check on the database's integrity. We will aim to reproduce the conceptual relational schema that we designed in coursework 1 and apply constraints to the database which conform with the assumptions that we previously made.

Relational Schema

Here is our schema from Coursework 1 (we have underlined primary keys and italicised foreign keys):

Student (Student_ID, S_Surname, S_Given_Name, *Email*, S_Start_Date, S_Finish_Date)
Staff (Staff_ID, ST_Surname, ST_Given_Name, ST_Start_Date, ST_Finish_Date, *Email*)
Member (*Email*, Resource_Cap)
Resource (Resource ID, *ISBN/EAN*, Location, #th_copy)
Resource Information (ISBN/EAN, *Class_Number*, Resource_Type, Title, Edition, Publication_Year, Loan_Period)
Author (Author_ID, A_Given_Name, A_Surname)
Resource–Author (ISBN/EAN, Author_ID)
Publisher (Publisher_ID, Publisher)
Resource-Publisher (ISBN/EAN, *Publisher_ID*)
Class (*Class_Number*, Class_Name)
Loan (Loan_Ref#, *Email*, *Resource_ID*, Issue_Date, Return_Date)
Fine (*Loan_Ref#*, Date_Paid, *Email*)

When setting out to implement this schema in SQL, we had to make some modifications to the relational schema plan. We changed our schema to the following:

MEMBERSHIP (EMAIL, RCAP, MNAME, ID, SD, FD, MTYPE, SSTATUS)
CLASSES (*CNUMBER*, CNAME)
RESOURCEINFORMATION (*ISBNEAN*, *CNUMBER*, RTYPE, TITLE, EDITION, PYEAR, LPERIOD)
RESOURCES (*RID*, *ISBNEAN*, LOCATION, THCOPY)
AUTHOR (*AID*, ANAME)
RESOURCEAUTHOR (*ISBNEAN*, *AID*)
PUBLISHER (*PID*, PNAME)
RESOURCEPUBLISHER (*ISBNEAN*, *PID*)
LOAN (*LREF*, *EMAIL*, *RID*, IDATE, RDATE)
FINE (*LREF*, DPAID, *EMAIL*)

Following the feedback given to us from the first half of the coursework highlighting the need to simplify certain aspects of the database, we decided to combine the tables Student, Staff, Member into one table called MEMBERSHIP with the extra columns MTYPE and SSTATUS. MTYPE is used to identify whether the member was a Student or Staff. SSTATUS is the suspension status of a member, the decision to add this column is that a member might be suspended for reasons other than owing too large a fine amount.

We have renamed some of our columns for the sake of simplicity. We have also capitalised our table and column names for clarity.

Creating Tables
In this section, we will show the SQL statements used to create the tables in the database, briefly outlining the constraints enforced. The tables must be created in the following order because of foreign key constraints- which mean referenced tables have to be created before the tables that reference them.

MEMBERSHIP
The alter statement at the start ensures that the date format for our database stays consistent at dd-mm-yyyy.
Email is the primary key constraint for the Member table. Any value in the Email column is therefore both necessarily unique - differing from all the other values contained within this attribute, and not null - this attribute must contain a value for a tuple to exist. We have imposed two checks on the Email attribute in order to enforce the data's validity- these checks help to make sure that a '@' and '.' are present in the string.

We have imposed a not null constraint on every attribute in the table. We have restricted the number of characters to 55 for EMAIL since a member's full name may be included in their email, and 40 for MNAME the table as this avoids storage of non-sensical values.

The data type of ID means that only IDs that are 10 digits long can be present in this column.

A member's start and end dates at university are necessary to manage a student's library membership. We have therefore imposed a 'not null' constraint on SD and FD, meaning that the SD and FD columns must contain a value for a tuple to exist. We have also instigated a check constraint requiring that FD is greater than SD since a member must finish at university after they begin.

A constraint has also been added to ensure that if a member is a student then their resource cap is 5, and if they are staff their resource cap is 10.

```
ALTER SESSION SET NLS_DATE_FORMAT = 'dd-mm-yyyy';
CREATE TABLE MEMBERSHIP (
EMAIL VARCHAR2(55) CONSTRAINT PK_MEMBERSHIP PRIMARY KEY,
RCAP NUMBER(2) NOT NULL,
MNAME VARCHAR2(40),
ID NUMBER(10),
SD DATE NOT NULL,
FD DATE NOT NULL,
MTYPE VARCHAR2(7) NOT NULL,
SSTATUS VARCHAR2(1) NOT NULL,
CHECK (LENGTH(ID)=10),
CHECK (SD < FD),
CHECK ((MTYPE = 'Student' AND RCAP = 5) OR (MTYPE = 'Staff' AND RCAP =
10)) ,
CHECK (SSTATUS IN ('Y', 'N')),
CHECK (EMAIL LIKE '%@%'),
CHECK (EMAIL LIKE '%.%')
);
```

CLASSES
CNUMBER has been given the primary key constraint for the CLASS table. We set class number (CNUMBER) to be a 4-digit-long number that allows there to be up to 9999 classes allocated to up to 9999 subjects. Also, class name is a string (CNAME) which is set to have a maximum of 30 characters, and which can contain enough information to specify a class name (including shorthand references)- this avoids the storage of non-essential information.

```
CREATE TABLE CLASSES (
CNUMBER NUMBER(4)
CONSTRAINT PK_CLASS PRIMARY KEY,
CNAME VARCHAR2(30) NOT NULL
);
```

RESOURCEINFORMATION
ISBNEAN has been given the primary key constraint for the RESOURCEINFORMATION table. It is set to be a number containing up to 13 digits. We have imposed a check constraint on ISBNEAN to check that the length is between 10 and 13 digits. We have set this limit to accommodate the varying lengths of ISBNs and EANs. ISBNs were 10 digits long until December 2006 but since 2007 are 13 digits long. EANs are typically 13 digits long. Class number (CNUMBER) represents the class of the resource and should be not null. It is set as the foreign key linking the RESOURCEINFORMATION table to the CLASSES table. The resource type (RTYPE) and resource title (TITLE) are set to be 10 characters long and 100 characters long respectively. The RTYPE is restricted to 10 characters since this saves storage space as the resource type can easily be indicated. Values for RTYPE and TITLE cannot be null as a resource's title and type form part of its essential description. Furthermore, there is a constraint to check RTYPE is one of the following: CD, DVD, Book, Video and Journal. Edition is a number with a maximum of 2 digits since there are rarely many editions of the same resource. Year of publication (PYEAR) is 4-digit number used to store the year in which the resource was published. Loan period (LPERIOD) is a number with a maximum of 2 digits showing the number of days a resource can be borrowed out for- this number depends on what the resource is.

```
CREATE TABLE RESOURCEINFORMATION (
ISBNEAN NUMBER(13) CONSTRAINT PK_RESOURCEINFORMATION PRIMARY
KEY,
CONSTRAINT CK_ISBNEAN
CHECK ((LENGTH(ISBNEAN)>=10) AND (LENGTH(ISBNEAN)<=13)),
CNUMBER NUMBER(4) NOT NULL,
CONSTRAINT FK_RESOURCEINFORMATION_CNUMBER FOREIGN
KEY(CNUMBER)
REFERENCES CLASSES,
RTYPE VARCHAR2(10) NOT NULL,
TITLE VARCHAR2(100) NOT NULL,
EDITION NUMBER(2),
PYEAR NUMBER(4),
LPERIOD NUMBER(2) NOT NULL,
CONSTRAINT CK_RTYPE
```

```
CHECK ((RTYPE IN ('CD', 'DVD', 'Book', 'Video', 'Journal')))
);
```

RESOURCES

RID has been given the primary key constraint for the RESOURCES table. The RID is generated automatically using IDENTITY. RID is set with a maximum of 10 digits and can store at most 9999999999 resources- this is more than enough to supply a sizeable college library. ISBNEAN is set to be NOT NULL and is a foreign key connecting the RESOURCE INFORMATION table to the RESOURCES table. LOCATION is set to be a 4-character string, showing the situation of a resource- the first 2 digits show the floor location, and the last 2 digits show the shelf position (as explained in coursework 1). THCOPY is the number identifying the copy of the resource which is set to be at most 2 digits long, allowing the resource to have a maximum of 99 copies. It cannot be a null value as there must be at least one copy in the library if it is to be worth storing information about a resource.

```
CREATE TABLE RESOURCES (
RID NUMBER(10)
GENERATED BY DEFAULT ON NULL AS IDENTITY
CONSTRAINT PK_RESOURCES PRIMARY KEY,
ISBNEAN NUMBER(13) NOT NULL,
CONSTRAINT FK_RESOURCES_ISBNEAN FOREIGN KEY(ISBNEAN)
REFERENCES RESOURCEINFORMATION,
LOCATION VARCHAR2(4) NOT NULL,
THCOPY NUMBER(2) NOT NULL
);
```

AUTHOR

AID has been given the primary key constraint for the AUTHOR table. The AID is generated automatically using IDENTITY. It is set to contain at most 5 digits. ANAME is author name which is a maximumly 80-character long string. It is long enough to store people names or companies' names if the author is a company or organisation. It cannot be a null value.

```
CREATE TABLE AUTHOR (
AID NUMBER(5)
GENERATED BY DEFAULT ON NULL AS IDENTITY
CONSTRAINT PK_AUTHOR PRIMARY KEY,
ANAME VARCHAR2(80) NOT NULL
);
```

RESOURCEAUTHOR

ISBNEAN and AID are set as the composite primary key for the RESOURCEAUTHOR table. ISBNEAN is a foreign key which creates a link to the table RESOURCEINFORMATION. AID is a foreign key which creates a link to the table AUTHOR.

```
CREATE TABLE RESOURCEAUTHOR (
ISBNEAN NUMBER(13),
AID NUMBER(5),
CONSTRAINT PK_RESOURCEAUTHOR PRIMARY KEY(ISBNEAN,AID),
CONSTRAINT FK_RESOURCEAUTHOR_ISBNEAN FOREIGN KEY(ISBNEAN)
REFERENCES RESOURCEINFORMATION,
CONSTRAINT FK_RESOURCEAUTHOR_AID FOREIGN KEY(AID)
REFERENCES AUTHOR,
CHECK ((LENGTH(ISBNEAN)>=10) AND (LENGTH(ISBNEAN)<=13))
);
```

PUBLISHER

PID has been given the primary key constraint for the PUBLISHER table. The PID is generated automatically using IDENTITY. It is set to be a number with a maximum of 5 digits. PNAME holds the name of the publisher. The datatype for PNAME is a string of up to 80 characters. This is long enough to store most companies' or organisations' names. This value cannot be null (must contain a value).

```
CREATE TABLE PUBLISHER (
PID NUMBER(5)
GENERATED BY DEFAULT ON NULL AS IDENTITY
CONSTRAINT PK_PUBLISHER PRIMARY KEY,
PNAME VARCHAR2(80) NOT NULL
);
```

RESOURCEPUBLISHER

ISBNEAN and PID are set as the composite primary key for the RESOURCEPUBLISHER table. ISBNEAN is a foreign key which links the table RESOURCEPUBLISHER to the table RESOURCEINFORMATION. PID is a foreign key linking RESOURCEPUBLISHER to the table PUBLISHER. We have imposed a check constraint as in RESOURCEINFORMATION to make sure that ISBNEAN is between 10 and 13 (greater than or equal to 10 and smaller than or equal to 13). This check is reinforced by the 13-digit limit set on ISBNEAN.

```
CREATE TABLE RESOURCEPUBLISHER (
ISBNEAN NUMBER(13),
PID NUMBER(5),
CONSTRAINT PK_RESOURCEPUBLISHER PRIMARY KEY(ISBNEAN,PID),
CONSTRAINT FK_RESOURCEPUBLISHER_ISBNEAN FOREIGN KEY(ISBNEAN)
REFERENCES RESOURCEINFORMATION,
CONSTRAINT FK_RESOURCEPUBLISHER_PID FOREIGN KEY(PID)
REFERENCES PUBLISHER,
CONSTRAINT CK_ISBNEAN3
CHECK ((LENGTH(ISBNEAN)>=10) AND (LENGTH(ISBNEAN)<=13))
);
```

LOAN

LREF has been given the primary key constraint for the LOAN table. The LREF is generated automatically using IDENTITY. LREF is a number which can contain at

most 10 digits and accommodate up to 9999999999 rows of loan information. EMAIL is a foreign key which links to MEMBERSHIP table, and RID is another foreign key which connects to RESOURCES table. IDATE is set to be date which records the date when the loan is issued, and it cannot be NULL. RDATE is set as the date the resource being returned, it can be NULL since information may be stored about a resource that has been taken out but not yet returned. There is a constraint which imposes the constraint that RDATE must be later than IDATE – this helps to avoid the entry of erroneous dates and reflects the fact that a resource cannot be returned before it has been issued.

```
CREATE TABLE LOAN (
LREF NUMBER(10)
GENERATED BY DEFAULT ON NULL AS IDENTITY
CONSTRAINT PK_LOAN PRIMARY KEY,
EMAIL VARCHAR2(55),
CONSTRAINT FK_LOAN_LREF FOREIGN KEY(EMAIL)
REFERENCES MEMBERSHIP,
RID NUMBER(10),
CONSTRAINT FK_LOAN_RID FOREIGN KEY(RID)
REFERENCES RESOURCES,
RDATE DATE,
IDATE DATE NOT NULL,
CHECK (RDATE > IDATE)
);
```

FINE

LREF has been given the primary key constraint for the FINE table and is also a foreign key which links the FINE table to the LOAN table. The datatype of DPAID is a date and DPAID records the date on which a fine is paid- a value which can be null if the fine has not yet been paid. EMAIL is another foreign key which links the FINE table to the MEMBERSHIP table.

```
CREATE TABLE FINE (
LREF NUMBER(10) CONSTRAINT PK_FINE PRIMARY KEY,
CONSTRAINT FK_FINE_LREF FOREIGN KEY(LREF)
REFERENCES LOAN,
DPAID DATE,
EMAIL VARCHAR2(55),
CONSTRAINT FK_FINE_EMAIL FOREIGN KEY(EMAIL)
REFERENCES MEMBERSHIP
);
```

EVALUATIONS_LOG

Log_date is date value to record when the date of log happened. Action is a string to record the log action of “insert, update, or delete”.

```
CREATE TABLE EVALUATIONS_LOG (
log_date DATE,
action VARCHAR2(50)
```

);

Additional constraints
Prior to adding additional information to the database, triggers must be created as a check on the database's correctness. We can insert the data about suspended members with the value 'N' for SSTATUS and then update rows for suspended members to change their SSTATUS value to 'Y'. This makes it possible to add rows into the table for members who are currently suspended from the library.

Triggers

LOAN_START Trigger
This trigger will fire when someone or some system tries to INSERT a row into the LOAN table. We are creating this under the assumption that when someone tries to borrow a resource, the system will try to create an INSERT statement for the LOAN table. The trigger will then check if the resource you want to loan is available, if the borrower is suspended and if the borrower has reached the resource cap.

```
CREATE OR REPLACE TRIGGER LOAN_START
   BEFORE INSERT
   ON LOAN
   FOR EACH ROW
DECLARE
   L_CNT NUMBER(2);
   L_CNT2 NUMBER(2);
BEGIN
   SELECT RESOURCECANBORROW
      INTO L_CNT2
      FROM (SELECT MEMBERSHIP.MNAME, MEMBERSHIP.EMAIL,
   (CASE
   WHEN SSTATUS = 'N' AND MEMBERSHIP.FD > SYSDATE THEN
NVL(TABLE100.RESOURCECANBORROW1, MEMBERSHIP.RCAP)
   WHEN SSTATUS = 'Y' OR MEMBERSHIP.FD < SYSDATE THEN 0
   END)  AS RESOURCECANBORROW
   FROM (SELECT LOAN.EMAIL,
   MEMBERSHIP.RCAP - COUNT(CASE WHEN RDATE IS NULL THEN 1  END)
AS RESOURCECANBORROW1
   FROM LOAN
   INNER JOIN MEMBERSHIP
   ON LOAN.EMAIL = MEMBERSHIP.EMAIL
   GROUP BY LOAN.EMAIL,MEMBERSHIP.RCAP) TABLE100
   FULL OUTER JOIN MEMBERSHIP
   ON TABLE100.EMAIL = MEMBERSHIP.EMAIL) WHERE :NEW.EMAIL = EMAIL;
   SELECT COUNT(*)
      INTO L_CNT
      FROM (SELECT RID FROM (SELECT TABLE10.* FROM (SELECT
RESOURCES.* , RESOURCEINFORMATION.TITLE FROM
RESOURCEINFORMATION
```

```
INNER JOIN RESOURCES
ON RESOURCES.ISBNEAN = RESOURCEINFORMATION.ISBNEAN) TABLE10
INNER JOIN
(SELECT RID FROM RESOURCES
MINUS
SELECT RID FROM
(SELECT RID FROM LOAN WHERE RDATE IS NULL
)) TABLE20
ON TABLE10.RID = TABLE20.RID) WHERE :NEW.RID = RID);
   IF L_CNT = 0  THEN
      RAISE_APPLICATION_ERROR(-20101, 'The resource has already been
loaned out');
   END IF;
   IF L_CNT2 = 0 THEN
      RAISE_APPLICATION_ERROR(-20102, 'The user can"t borrow anymore
resources');
   END IF;
END;
```

Trigger to update the CNUMBER in RESOURCEINFORMATION when CNUMBER in CLASSES has been updated

```
CREATE OR REPLACE TRIGGER CLASSES_CASSCADE
BEFORE UPDATE OF CNUMBER
ON CLASSES
FOR EACH ROW
BEGIN
UPDATE RESOURCEINFORMATION SET CNUMBER = :NEW.CNUMBER
WHERE CNUMBER = :OLD.CNUMBER;
END;
```

Trigger to update the AID in RESOURCEAUTHOR when AID in AUTHOR has been updated

```
CREATE OR REPLACE TRIGGER RESOURCEAUTHOR_CASSCADE
BEFORE UPDATE OF AID
ON AUTHOR
FOR EACH ROW
BEGIN
UPDATE RESOURCEAUTHOR SET AID = :NEW.AID WHERE AID = :OLD.AID;
END;
```

Trigger to update the PID in RESOURCEPUBLISHER when PID in PUBLISHER has been updated

```
CREATE OR REPLACE TRIGGER RESOURCEPUBLISHER_CASSCADE
BEFORE UPDATE OF PID
ON PUBLISHER
FOR EACH ROW
```

```
BEGIN
UPDATE RESOURCEPUBLISHER SET PID = :NEW.PID WHERE PID = :OLD.PID;
END;
```

EVAL_CHANGE_TRIGGER

This trigger is used to log any table changes for the MEMBERSHIP table and keep track of updates to the system for accountability, error-tracking and other data security purposes. This trigger keeps track of modifications to the database- including whether data has been inserted, updated or deleted and the date of the change. We could create a trigger for every table to keep account of changes and the times they were made at but have only included this trigger as an example.

```
CREATE OR REPLACE TRIGGER EVAL_CHANGE_TRIGGER
AFTER INSERT OR UPDATE OR DELETE
ON MEMBERSHIP
DECLARE
log_action  EVALUATIONS_LOG.action%TYPE;
BEGIN IF INSERTING THEN    log_action := 'Insert';
ELSIF UPDATING THEN    log_action := 'Update';
ELSIF DELETING THEN    log_action := 'Delete';
ELSE    DBMS_OUTPUT.PUT_LINE('This code is not reachable.');
END IF;
INSERT INTO EVALUATIONS_LOG (log_date, action)
VALUES (SYSDATE, log_action);
END;
```

Here is a list of automated jobs that could be done to improve functionality of the library database system:

- A trigger to insert or update the value of days overdue for a resource in the FINE table. Our research informed us that this is not done by a trigger but by an automated scheduled job which runs every day because a resource is only overdue when the due date is before the system date (i.e., using DBMS_JOB or DBMS_SCHEDULER).

- There could be a trigger to change suspension status in MEMBERSHIP when a member's fine becomes 10 or more dollars (this is in line with the assignment brief's requirements). The fine amount should use a scheduled job to increase by 1 dollar a day if the fine is not paid. Once the fine amount is updated, the two views which calculate fines to get the latest suspension status can be updated by the automated job. Once this is done, the suspension status will be updated.

Data

This section will show how the sample data was created, how it's supposed to behave and will test the integrity of the database.
The INSERT Statements were made before the Triggers were implemented. This decision was made as with the Triggers running, we wouldn't be able to create the sample data that we wanted.

MEMBERSHIP

EMAIL	RCAP	MNAME	ID	SD	FD	MTYPE	SSTATUS
michelle.emmanuel@college.ac.uk	10	Michelle Emmanuel	1000000007	01-09-2017	01-09-2024	Staff	N
louis.edwards@college.ac.uk	10	Louis Edwards	1000000320	01-09-2017	01-09-2021	Staff	Y
sh354@college.ac.uk	5	Shamia Hassan	2017433231	01-09-2017	30-06-2021	Student	Y
na021@college.ac.uk	5	Nazia Ali	2017445231	01-09-2017	30-06-2021	Student	N
js354@college.ac.uk	5	Joe Smith	2019433231	01-09-2019	30-06-2022	Student	N
pf112@college.ac.uk	5	Paula Fowler	2019433233	01-09-2019	30-06-2022	Student	N
si.cheng@college.ac.uk	10	Si Cheng	1463919229	01-09-2020	01-09-2022	Staff	N
mc081@college.ac.uk	5	Matilda Clarke	2020994332	01-09-2020	30-06-2023	Student	N

The data above will be the sample data for our MEMBERSHIP table. The INSERT Script for the table will be in the Script.docx file.
To test the constraints of the table, we have used the INSERT statements below. The statements are written with the assumption that the sample data has already been inserted.

This statement has an Email (Primary key) that has already been inserted:

```
INSERT INTO MEMBERSHIP
VALUES('michelle.emmanuel@college.ac.uk',10,'Emmanuel',1002340007,
TO_DATE('01-10-2017', 'DD-MM-YYYY'), TO_DATE('01-12-2019', 'DD-MM-YYYY'),
'Staff', 'N');
```

The Start date is set after the Finishing date:

```
INSERT INTO MEMBERSHIP
VALUES('Phu.Pham@college.ac.uk',10,'Phu',1024540007, TO_DATE('01-10-2020',
'DD-MM-YYYY'), TO_DATE('01-12-2019', 'DD-MM-YYYY'), 'Staff', 'N');
```

The Member type has been set to an unrecognized type:

INSERT INTO MEMBERSHIP
VALUES('Phu.Pham@college.ac.uk',10,'Phu',1002322207, TO_DATE('01-10-2017', 'DD-MM-YYYY'), TO_DATE('01-12-2019', 'DD-MM-YYYY'), 'Boss', 'N');

The Email does fit the right format (no @):
INSERT INTO MEMBERSHIP
VALUES('Phu.Phamcollege.ac.uk',10,'Phu',1002322207, TO_DATE('01-10-2017', 'DD-MM-YYYY'), TO_DATE('01-12-2019', 'DD-MM-YYYY'), 'Staff', 'N');

The Resource cap is not 5 or 10:
INSERT INTO MEMBERSHIP
VALUES('Phu.Pham@college.ac.uk',12,'Phu',1002322207, TO_DATE('01-10-2017', 'DD-MM-YYYY'), TO_DATE('01-12-2019', 'DD-MM-YYYY'), 'Staff', 'N');

The member type is Student but the Resource cap was set to 10:
INSERT INTO MEMBERSHIP
VALUES('Phu.Pham@college.ac.uk',10,'Phu',1002322207, TO_DATE('01-10-2017', 'DD-MM-YYYY'), TO_DATE('01-12-2019', 'DD-MM-YYYY'), 'Student', 'N');

CLASSES

CNUMBER	CNAME
1	Computer Science
2	History
3	Foreign Language Study
4	Philosophy
5	Economy
6	Medical Science

The data above will be the sample data for our CLASSES table. The INSERT Script for the table will be in the Script.docx file.
To test the constraints of the table, we have used the INSERT statements below. The statements are written with the assumption that the sample data has already been inserted.

The statement has a Class number (Primary key) that has already been inserted:
INSERT INTO CLASSES VALUES(0001,'Finance');

This statement will run without triggering the Child – Parent error as the CLASSES_CASSCADE trigger would have handled the error:
UPDATE CLASSES SET CNUMBER = 15 WHERE CNAME = 'Foreign Language Study';

RESOURCEINFORMATION

ISBNEAN	CNUMBER	RTYPE	TITLE	EDITION	PYEAR	LPERIOD
9781292061184	1	Book	Database Systems	6	2010	14

9780198228589	2	Book	British history, 1815-1906	1	1991	14
9781907838101	3	CD	Chinese in steps. Student book 1	1	2011	2
9787560076195	3	Book	Little Oxford English-Chinese Dictionary	9	2015	14
9780415337977	4	Book	Philosophy : basic readings	2	2005	14
9770013061237	5	Journal	The Economist Issue NOVEMBER 2020	1	2020	0
9780472001095	6	Book	Virus	1	1959	14

The data above will be the sample data for our RESOURCEINFORMATION table. The INSERT Script for the table will be in the Script.docx file.
To test the constraints of the table, we have used the INSERT statements below. The statements are written with the assumption that the sample data has already been inserted.

The statement has an ISBNEAN (Primary key) that has already been inserted:
INSERT INTO RESOURCEINFORMATION
VALUES(9781292061184,0021,'Book','New idea',7,2012,14);

The ISBNEAN length is longer than the allowed length:
INSERT INTO RESOURCEINFORMATION
VALUES(97812920777777761184,0001,'Book','Database Systems',6,2010,14);

The CNUMBER (Foreign key) is trying to link to a CNUMBER that doesn't exist in the CLASSES table:
INSERT INTO RESOURCEINFORMATION
VALUES(9781292041184,0022,'Book','Database Systems',6,2010,14);

The Resource type 'Flier' is not of the allowed types of resource:
INSERT INTO RESOURCEINFORMATION
VALUES(9781117061184,0001,'Flier','Database Systems',6,2010,14);

RESOURCE

RID	ISBNEAN	LOCATION	THCOPY
1	9781292061184	0012	1
2	9781292061184	0012	2
3	9781292061184	0012	3
4	9780198228589	1120	1
5	9781907838101	0013	1
6	9787560076195	0017	1
7	9780415337977	0095	1
8	9770013061237	0031	1
9	9780472001095	0032	1
10	9780472001095	0032	2
11	9780472001095	0032	3
12	9780472001095	0032	4

The data above will be the sample data for our RESOURCE table. The INSERT Script for the table will be in the Script.docx file.
To test the constraints of the table, we have used the INSERT statements below. The Primary key RID is generated automatically but is still tested for the situation where RID data is manually inserted.
The statements are written with the assumption that the sample data has already been inserted.

The ISBNEAN (Foreign key) is trying to link to a ISBNEAN that doesn't exist in the RESOURCEINFORMATION table:
INSERT INTO RESOURCES (ISBNEAN, LOCATION, THCOPY) VALUES
(0123598235901, 0012, 3);

The statement has an RID (Primary key) that has already been inserted:
INSERT INTO RESOURCES (RID, ISBNEAN, LOCATION, THCOPY) VALUES
(2, 9780472001095, 0013, 4);

AUTHOR

AID	ANAME
1	Thomas Connolly
2	Carolyn Begg
3	Norman McCord
4	George Zhang
5	Linda Li
6	Lik Suen
7	Oxford University Press
8	Nigel Warburton
9	The Economist
10	Wolfhard Weidel

The data above will be the sample data for our AUTHOR table. The INSERT Script for the table will be in the Script.docx file.
To test the constraints of the table, we have used the INSERT statements below. The Primary key AID is generated automatically but is still tested for in the situation where AID data is manually inserted.

The statement has an AID (Primary key) that has already been inserted:
INSERT INTO AUTHOR VALUES
(6, ' Sandi Toksvig ');

RESOURCEAUTHOR

ISBNEAN	AID
9770013061237	9
9780198228589	3
9780415337977	8
9780472001095	10
9781292061184	1
9781292061184	2

9781907838101	4
9781907838101	5
9781907838101	6
9787560076195	7

The data above will be the sample data for our RESOURCEAUTHOR table. The INSERT Script for the table will be in the Script.docx file.
To test the constraints of the table, we have used the INSERT statements below. The statements are written with the assumption that the sample data has already been inserted

The AID (Foreign key) is trying to link to a AID that doesn't exist in the AUTHOR table:
INSERT INTO RESOURCEAUTHOR VALUES (9781292061184, 15);
The ISBNEAN (Foreign key) is trying to link to a ISBNEAN that doesn't exist in the RESOURCEINFORMATION table:
INSERT INTO RESOURCEAUTHOR VALUES (9781292061100, 7);
The statement has an (ISBNEAN, AID)(Composite primary key) that has already been inserted:
INSERT INTO RESOURCEAUTHOR VALUES (9781292061184, 1);
The ISBNEAN length is longer than the allowed length:
INSERT INTO RESOURCEAUTHOR VALUES (97812920611849, 1);
The ISBNEAN length is shorter than the allowed length:
INSERT INTO RESOURCEAUTHOR VALUES (978129209, 1);

PUBLISHER

PID	PNAME
1	Pearson
2	Oxford University Press
3	Sinolingua London Ltd.
4	Foreign Language Teaching and Research Press
5	Routledge
6	The Economist Group
7	University of Michigan Press

The data above will be the sample data for our PUBLISHER table. The INSERT Script for the table will be in the Script.docx file.
To test the constraints of the table, we have used the INSERT statements below. The Primary key PID is generated automatically but is still tested for in the situation where PID data is manually inserted.

The statement has an AID (Primary key) that has already been inserted:
INSERT INTO PUBLISHER VALUES (3, 'Cambridge University Press');

RESOURCEPUBLISHER

ISBNEAN	PID
9770013061237	6
9780198228589	2
9780415337977	5
9780472001095	7

9781292061184	1
9781907838101	3
9787560076195	2
9787560076195	4

The data above will be the sample data for our RESOURCEPUBLISHER table. The INSERT Script for the table will be in the Script.docx file.
To test the constraints of the table, we have used the INSERT statements below. The statements are written with the assumption that the sample data has already been inserted

The statement has an (ISBNEAN, PID)(Composite primary key) that has already been inserted:
INSERT INTO RESOURCEPUBLISHER VALUES (9781292061184, 1);

The ISBNEAN (Foreign key) is trying to link to a ISBNEAN that doesn't exist in the RESOURCEINFORMATION table:
INSERT INTO RESOURCEPUBLISHER VALUES (9770000000000,1);

The PID (Foreign key) is trying to link to a PID that doesn't exist in the PUBLISHER table:
INSERT INTO RESOURCEPUBLISHER VALUES (9770013061237,1996);

The ISBNEAN length is longer than the allowed length:
INSERT INTO RESOURCEPUBLISHER VALUES (977001306123732894858,1);

The ISBNEAN length is shorter than the allowed length:
INSERT INTO RESOURCEPUBLISHER VALUES (97794858,1);

LOAN

LREF	EMAIL	RID	RDATE	IDATE
1	michelle.emmanuel@college.ac.uk	1	17-09-2020	01-09-2020
2	michelle.emmanuel@college.ac.uk	10	03-09-2020	01-09-2020
3	sh354@college.ac.uk	2	05-09-2020	03-09-2020
4	na021@college.ac.uk	5	09-09-2020	04-09-2020
5	mc081@college.ac.uk	9	09-09-2020	05-09-2020
6	mc081@college.ac.uk	5	12-09-2020	10-09-2020
7	louis.edwards@college.ac.uk	11	-	07-11-2020
8	louis.edwards@college.ac.uk	1	20-11-2020	07-11-2020
9	louis.edwards@college.ac.uk	6	20-11-2020	07-11-2020
10	louis.edwards@college.ac.uk	7	-	07-11-2020
11	na021@college.ac.uk	9	30-11-2020	10-11-2020

12	na021@college.ac.uk	5	17-11-2020	15-11-2020
13	sh354@college.ac.uk	3	20-11-2020	16-11-2020
14	sh354@college.ac.uk	5	-	01-12-2020

The data above will be the sample data for our LOAN table. The INSERT Script for the table will be in the Script.docx file.
To test the constraints of the table, we have used the INSERT statements below. The Primary key LREF is generated automatically but is still tested for in the situation where PID data is manually inserted.

The statement has an LREF (Primary key) that has already been inserted:
INSERT INTO LOAN VALUES (3,'michelle.emmanuel@college.ac.uk', 1, TO_DATE('21-11-2020', 'DD-MM-YYYY'), TO_DATE('25-11-2020', 'DD-MM-YYYY'));

The EMAIL (Foreign key) is trying to link to an EMAIL that doesn't exist in the MEMBERSHIP table:
INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('test.email@college.ac.uk', 1, TO_DATE('01-09-2020', 'DD-MM-YYYY'), TO_DATE('17-09-2020', 'DD-MM-YYYY'));

The RID (Foreign key) is trying to link to a RID that doesn't exist in the RESOURCES table:
INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('michelle.emmanuel@college.ac.uk', 1996, TO_DATE('01-09-2020', 'DD-MM-YYYY'), TO_DATE('17-09-2020', 'DD-MM-YYYY'));

The Issue date is set after the Return date:
INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('michelle.emmanuel@college.ac.uk', 1, TO_DATE('01-12-2020', 'DD-MM-YYYY'), TO_DATE('21-11-2020', 'DD-MM-YYYY'));

The Resource is not available to borrow (The RID 7 is still loaned out):
INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('michelle.emmanuel@college.ac.uk', 7, TO_DATE('01-12-2020', 'DD-MM-YYYY'), NULL);

The Borrower is suspended (Shamia Hassan is Suspended):
INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('sh354@college.ac.uk', 1, TO_DATE('01-12-2020', 'DD-MM-YYYY'), NULL);

The Borrower has reached the Resource cap (The sixth INSERT Statement will fail as a Student can't borrow more than 5 resource at a time):
INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('na021@college.ac.uk', 1, TO_DATE('01-12-2020', 'DD-MM-YYYY'), NULL);
INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('na021@college.ac.uk', 2, TO_DATE('01-12-2020', 'DD-MM-YYYY'), NULL);
INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('na021@college.ac.uk', 3, TO_DATE('01-12-2020', 'DD-MM-YYYY'), NULL);
INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('na021@college.ac.uk', 4, TO_DATE('01-12-2020', 'DD-MM-YYYY'), NULL);

INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('na021@college.ac.uk', 6, TO_DATE('01-12-2020', 'DD-MM-YYYY'), NULL);
INSERT INTO LOAN (EMAIL,RID,IDATE,RDATE) VALUES ('na021@college.ac.uk', 10, TO_DATE('01-12-2020', 'DD-MM-YYYY'), NULL);

FINE

LREF	DPAID	EMAIL
1	17-09-2020	michelle.emmanuel@college.ac.uk
4	09-09-2020	na021@college.ac.uk
7	-	louis.edwards@college.ac.uk
10	-	louis.edwards@college.ac.uk
11	02-12-2020	na021@college.ac.uk
14	-	sh354@college.ac.uk

The data above will be the sample data for our FINE table. The INSERT Script for the table will be in the Script.docx file.
To test the constraints of the table, we have used the INSERT statements below. The statements are written with the assumption that the sample data has already been inserted

The statement has an LREF (Primary key) that has already been inserted:
INSERT INTO FINE VALUES (4, TO_DATE('15-09-2020', 'DD-MM-YYYY'), 'louis.edwards@college.ac.uk');

The LREF (Foreign key) is trying to link to a LREF that doesn't exist in the LOAN table:
INSERT INTO FINE VALUES (1997, TO_DATE('09-09-2020', 'DD-MM-YYYY'), 'na021@college.ac.uk');

The EMAIL (Foreign key) is trying to link to a EMAIL that doesn't exist in the MEMBERSHIP table:
INSERT INTO FINE VALUES (5, TO_DATE('09-09-2020', 'DD-MM-YYYY'), 'test.email@college.ac.uk');

Views

For complex and common queries, views must be created to facilitate use of the database. This section shows these views, their purpose and their intended results.

POPULARLIST
This view aims to get the data on how popular a resource is in a specified time period. In this example the TIMESBORROWED shows the number of times a resource was loaned out from 09-2020 to 10-2020. This view is a necessity as one of the requirements of the database is to support a list of popular resources and any system that wants to do that need to go through this view.

```
CREATE VIEW POPULARLIST AS
(SELECT TITLE, COUNT(TITLE) AS TIMESBORROWED FROM
(SELECT RESOURCES.ISBNEAN, LOAN.*, RESOURCEINFORMATION.TITLE
FROM RESOURCES
INNER JOIN LOAN
ON RESOURCES.RID = LOAN.RID
INNER JOIN RESOURCEINFORMATION
ON RESOURCES.ISBNEAN = RESOURCEINFORMATION.ISBNEAN
)
WHERE IDATE BETWEEN TO_DATE('01-09-2020','dd-mm-yyyy') AND
TO_DATE('30-09-2020','dd-mm-yyyy')
GROUP BY TITLE
        );
```

TITLE	TIMESBORROWED
Chinese in steps. Student book 1	2
Database Systems	2
Virus	2

COPIESAVAILABLE

This view aims to show all the available copies of all resources. All rows in this table shows copies that can be loaned out and their information. This view is a fundament of the database as before any loans happen, the system needs to find out if the resource has a copy that is available for loan.

```
CREATE VIEW COPIESAVAILABLE AS
(SELECT TABLE1.* FROM (SELECT RESOURCES.* ,
RESOURCEINFORMATION.TITLE FROM RESOURCEINFORMATION
INNER JOIN RESOURCES
ON RESOURCES.ISBNEAN = RESOURCEINFORMATION.ISBNEAN) TABLE1
INNER JOIN
(SELECT RID FROM RESOURCES
MINUS
SELECT RID FROM
(SELECT RID FROM LOAN WHERE RDATE IS NULL
)) TABLE2
ON TABLE1.RID = TABLE2.RID
);
```

RID	ISBNEAN	LOCATION	THCOPY	TITLE
1	9781292061184	0012	1	Database Systems
2	9781292061184	0012	2	Database Systems
3	9781292061184	0012	3	Database Systems
4	9780198228589	1120	1	British history, 1815-1906

6	9787560076195	0017	1	Little Oxford English-Chinese Dictionary
8	9770013061237	0031	1	The Economist Issue NOVEMBER 2020
9	9780472001095	0032	1	Virus
10	9780472001095	0032	2	Virus
12	9780472001095	0032	4	Virus

FINESTATUS

This view shows the Fine amount that each Loan has occurred if it has become overdue. It shows both the outstanding fines and the ones that has been paid off. This is shown in the DPAID column with unpaid fines as NULL and paid fines with a date inserted. This is a vital view for the system to send out invoices and give out suspensions as needed.

```
CREATE OR REPLACE VIEW FINESTATUS AS
(SELECT FINEINFO.*, FINE.DPAID FROM
(SELECT EMAIL,LREF, IDATE, IDATE+LPERIOD AS DUEDATE, RDATEorTODAY,
RDATEorTODAY-(IDATE+LPERIOD) AS FINEAMOUNT
FROM (SELECT LOAN.LREF, LOAN.EMAIL, LOAN.IDATE,
TO_DATE(NVL(LOAN.RDATE,SYSDATE),' dd-mm-yyyy ') AS RDATEorTODAY,
RES1.*
FROM LOAN
INNER JOIN (SELECT RESOURCES.RID, RESOURCEINFORMATION.* FROM
RESOURCES INNER JOIN RESOURCEINFORMATION ON
RESOURCES.ISBNEAN = RESOURCEINFORMATION.ISBNEAN) RES1 ON
LOAN.RID = RES1.RID)
WHERE RDATEorTODAY-(IDATE+LPERIOD) >0) FINEINFO
INNER JOIN FINE
ON FINEINFO.LREF = FINE.LREF
);
```

EMAIL	LREF	IDATE	DUEDATE	RDATEORTODAY	FINEAMOUNT	DPAID
michelle.emmanuel@college.ac.uk	1	01-09-2020	15-09-2020	17-09-2020	2	17-09-2020
na021@college.ac.uk	4	04-09-2020	06-09-2020	09-09-2020	3	09-09-2020
louis.edwards@college.ac.uk	7	07-11-2020	21-11-2020	11-12-2020	20	-
louis.edwards@college.ac.uk	10	07-11-2020	21-11-2020	11-12-2020	20	-

na021@college.ac.uk	11	10-11-2020	24-11-2020	30-11-2020	6	02-12-2020
sh354@college.ac.uk	14	01-12-2020	03-12-2020	11-12-2020	8	-

TOTALFINENOTPAID
This view shows the total fine a Member has incurred and if they had incurred enough fines to be suspended. The SSTATUS column here shows if the Member should be suspended or not.

```
CREATE OR REPLACE VIEW TOTALFINENOTPAID AS
(SELECT EMAIL, SUM(FINEAMOUNT) AS TOTALFINE,
   CASE
      WHEN SUM(FINEAMOUNT) >= 10 THEN 'Y'
      WHEN SUM(FINEAMOUNT) <10 THEN 'N'
   END AS SSTATUS
FROM FINESTATUS
WHERE DPAID IS NULL
GROUP BY EMAIL
);
```

EMAIL	TOTALFINE	SSTATUS
sh354@college.ac.uk	8	N
louis.edwards@college.ac.uk	40	Y

Queries

Find a Member with a particular name
This is a query to find a member based on their name. This is a common query as the library system will need to look up a member using their name to find further information.

```
SELECT *
FROM MEMBERSHIP
WHERE (MNAME LIKE '%Edwards');
```

EMAIL	RCAP	MNAME	ID	SD	FD	MTYPE	SSTATUS
louis.edwards@college.ac.uk	10	Louis Edwards	1000000320	01-09-2017	01-09-2021	Staff	Y

Calculate the average length of a loan (DAYS)

This query can be used to estimate how long a Resource will be loaned out. It finds the average length of a loan taken out with the library. This information can help with forecasting and evaluating the effectiveness of the Library system. The data has been rounded to 2 decimal places.

```
SELECT  ROUND(AVG((RDATE - IDATE)), 2) AVGDAYS FROM LOAN WHERE
RDATE IS NOT NULL
```

AVGDAYS
7.55

Find the titles and resource types of the resources that are currently on loan, grouped by location.

```
SELECT NOTRETURN.RID, RESOURCEINFORMATION.TITLE,
RESOURCEINFORMATION.RTYPE, NOTRETURN.LOCATION
FROM (SELECT LOAN.RID, RESOURCES.ISBNEAN, RESOURCES.LOCATION
FROM LOAN
INNER JOIN RESOURCES ON LOAN.RID = RESOURCES.RID
WHERE RDATE IS NULL) NOTRETURN
INNER JOIN RESOURCEINFORMATION ON NOTRETURN.ISBNEAN =
RESOURCEINFORMATION.ISBNEAN
ORDER BY NOTRETURN.LOCATION
```

RID	TITLE	RTYPE	LOCATION
5	Chinese in steps. Student book 1	CD	0013
11	Virus	Book	0032
7	Philosophy : basic readings	Book	0095

Find the publisher name of the specific resource

This query shows the publisher's name for the resource 'Database Systems'. As the database system is not built to store multiple publishers as separate entities, this query is required to find the publisher information on a resource.

```
SELECT TITLE, ISBNEAN, PNAME
FROM
(SELECT RESOURCEINFORMATION.TITLE,
RESOURCEINFORMATION.ISBNEAN, PUBLISHER.PNAME
FROM RESOURCEINFORMATION
INNER JOIN RESOURCEPUBLISHER
ON RESOURCEINFORMATION.ISBNEAN = RESOURCEPUBLISHER.ISBNEAN
INNER JOIN PUBLISHER
ON RESOURCEPUBLISHER.PID = PUBLISHER.PID)
WHERE TITLE = 'Database Systems';
```

TITLE	ISBNEAN	PNAME
Database Systems	9781292061184	Pearson

Count the number of loans taken out in a specific time period
The query shows the number of loans taken out since November 2020. This query is useful in evaluating the effectiveness of the Library system as it could be used to show how busy the system is throughout the year.

```
SELECT COUNT(IDATE) TOTALNOLOANS
FROM LOAN
WHERE (IDATE >= TO_DATE('01-11-2020', 'DD-MM-YYYY'));
```

TOTALNOLOANS
8

Find the list of titles for resources that were loaned out in November 2020, ordered alphabetically by title.

```
SELECT DISTINCT TITLE, ISBNEAN
FROM RESOURCEINFORMATION
WHERE ISBNEAN IN
(SELECT ISBNEAN
FROM RESOURCES
JOIN LOAN
ON LOAN.RID = RESOURCES.RID
WHERE (IDATE <= TO_DATE('30-11-2020', 'DD-MM-YYYY')) AND (IDATE >=
TO_DATE('01-11-2020', 'DD-MM-YYYY')))
ORDER BY TITLE;
```

TITLE	ISBNEAN
Chinese in steps. Student book 1	9781907838101
Database Systems	9781292061184
Little Oxford English-Chinese Dictionary	9787560076195
Philosophy : basic readings	9780415337977
Virus	9780472001095

Find the member name and ID of members who have a current valid membership with the library
This query shows which person is currently holding a valid membership. This would help the Library system as to give that person the correct access level to the Library system.

```
SELECT MNAME, ID
FROM MEMBERSHIP
WHERE (FD >= SYSDATE) AND SSTATUS = 'N';
```

MNAME	ID
Michelle Emmanuel	1000000007
Nazia Ali	2017445231
Joe Smith	2019433231
Paula Fowler	2019433233

Si Cheng	1463919229
Matilda Clarke	2020994332

List the member names and IDs of members who have currently have a resource taken out with the library.

```
SELECT DISTINCT MEMBERSHIP.MNAME, MEMBERSHIP.ID
FROM MEMBERSHIP
INNER JOIN
(SELECT EMAIL
FROM LOAN
WHERE LOAN.RDATE IS NULL) MWITHLOAN
ON MEMBERSHIP.EMAIL = MWITHLOAN.EMAIL;
```

MNAME	ID
Louis Edwards	1000000320
Shamia Hassan	2017433231

Find the titles, ISBNs, years of publication and edition of resources written by the author 'Linda Li', ordered by year of publication

This is an important query as Library users would most likely want to search for resources by Author and this query facilitates that.

```
SELECT TITLE, ISBNEAN, PYEAR, EDITION
FROM
(SELECT RESOURCEINFORMATION.TITLE,
RESOURCEINFORMATION.ISBNEAN, RESOURCEINFORMATION.PYEAR,
RESOURCEINFORMATION.EDITION
FROM RESOURCEINFORMATION
INNER JOIN RESOURCEAUTHOR
ON RESOURCEINFORMATION.ISBNEAN = RESOURCEAUTHOR.ISBNEAN
INNER JOIN AUTHOR
ON RESOURCEAUTHOR.AID = AUTHOR.AID
WHERE ANAME = 'Linda Li')
ORDER BY PYEAR;
```

TITLE	ISBNEAN	PYEAR	EDITION
Chinese in steps. Student book 1	9781907838101	2011	1

List resource usage (history of a particular resource)

This query shows the entire loan history of a particular resource. The use cases for this query would be to track who has borrowed the resource and how popular the resource is.

```
SELECT RESOURCEINFORMATION.TITLE, LOAN.EMAIL, LOAN.IDATE,
LOAN.RDATE FROM LOAN
INNER JOIN RESOURCES
ON LOAN.RID = RESOURCES.RID
INNER JOIN RESOURCEINFORMATION
ON RESOURCES.ISBNEAN = RESOURCEINFORMATION.ISBNEAN
```

WHERE RESOURCEINFORMATION.TITLE = 'Database Systems';

TITLE	EMAIL	IDATE	RDATE
Database Systems	michelle.emmanuel@college.ac.uk	01-09-2020	17-09-2020
Database Systems	louis.edwards@college.ac.uk	07-11-2020	20-11-2020
Database Systems	sh354@college.ac.uk	03-09-2020	05-09-2020
Database Systems	sh354@college.ac.uk	16-11-2020	20-11-2020

Find the class numbers and names for all the titles loaned out since the beginning of 2020, ordered by class number

This query might be used by library administrators to find a list of all the class number and class names with their titles and redesign or reorder the class system used in the library. This query only selects the class numbers and names of resources that have been used in the last year since these resources are the most relevant.

```
SELECT DISTINCT TITLE, CNUMBER, CNAME
FROM
(SELECT RESOURCEINFORMATION.TITLE, CLASSES.CNUMBER,
CLASSES.CNAME
FROM
CLASSES
INNER JOIN RESOURCEINFORMATION
ON RESOURCEINFORMATION.CNUMBER = CLASSES.CNUMBER
INNER JOIN RESOURCES
ON RESOURCEINFORMATION.ISBNEAN = RESOURCES.ISBNEAN
INNER JOIN LOAN
ON LOAN.RID = RESOURCES.RID
WHERE (LOAN.IDATE <= SYSDATE) AND (LOAN.IDATE >= TO_DATE('01-01-
2020', 'DD-MM-YYYY')))
ORDER BY CNUMBER
;
```

TITLE	CNUMBER	CNAME
Database Systems	1	Computer Science
Chinese in steps. Student book 1	3	Foreign Language Study
Little Oxford English-Chinese Dictionary	3	Foreign Language Study
Philosophy : basic readings	4	Philosophy
Virus	6	Medical Science

Show the number of resources a member is currently able to borrow

The following query shows the number of resources a member can currently borrow. The query finds this by calculating the total number of resources the member has already taken out and deducts this sum from the member's allocated resource cap.

The query also takes in account whether the member has been suspended and returns the RESOURCECANBORROW column value as 0 if they are suspended.

```
SELECT MEMBERSHIP.MNAME, MEMBERSHIP.EMAIL,
(CASE
WHEN SSTATUS = 'N' AND MEMBERSHIP.FD > SYSDATE THEN
NVL(TABLE100.RESOURCECANBORROW1, MEMBERSHIP.RCAP)
WHEN SSTATUS = 'Y' OR MEMBERSHIP.FD < SYSDATE THEN 0
END)  AS RESOURCECANBORROW
FROM (SELECT LOAN.EMAIL,
MEMBERSHIP.RCAP - COUNT(CASE WHEN RDATE IS NULL THEN 1  END) AS
RESOURCECANBORROW1
FROM LOAN
INNER JOIN MEMBERSHIP
ON LOAN.EMAIL = MEMBERSHIP.EMAIL
GROUP BY LOAN.EMAIL,MEMBERSHIP.RCAP) TABLE100
FULL OUTER JOIN MEMBERSHIP
ON TABLE100.EMAIL = MEMBERSHIP.EMAIL;
```

MNAME	EMAIL	RESOURCECANBORROW
Louis Edwards	louis.edwards@college.ac.uk	0
Michelle Emmanuel	michelle.emmanuel@college.ac.uk	10
Shamia Hassan	sh354@college.ac.uk	0
Matilda Clarke	mc081@college.ac.uk	5
Nazia Ali	na021@college.ac.uk	5
Si Cheng	si.cheng@college.ac.uk	10
Joe Smith	js354@college.ac.uk	5
Paula Fowler	pf112@college.ac.uk	5

Database Security Considerations

In this section, we cover data security issues that would need to be addressed by administrators and users of our system. Oracle should be set up with database authentication, where every user is created with their own password to be provided when the user tries to make connection. This stops database access by unauthorised persons as the connection is refused for users with invalid passwords. Multi-factor authentication can be introduced to prevent unauthorised persons from logging in with valid passwords if they are leaked. The database administrator can set password lifetimes and expire default user accounts to reinforce security.
Journaling enables effective recovery after a system failure- a record of modifications to the database is kept with the chronological activity of system objects in a separate log file. Article 24 of the UK's Data Protection Act of 2018 (GDPR) also demands that organisations show security measures instigated in the interests of compliance.

Moreover, copies of the database (backups) need to be made regularly to save crucial files in the case of a system crash or hard drive failure. Offline storage media can be used to log files and programs. The SQL Server database has a transaction log which records changes made with every transaction and can be used in the case of a system failure. GDPR demands the prevention of the loss or alteration of PII data (*Data Protection Act* 2018). Protecting data from becoming invalid avoids inaccurate results. We have therefore created a table in our database which can be used to record types of changes to the database and the corresponding date to prevent accidental deletion or edits to data.

Setting up access controls ensure that employees only view relevant personal information. Recital 39 of the GDPR states that personal data needs to be processed to '[prevent] unauthorised access or use of personal data and the equipment used for processing' (*Data Protection Act* 2018). The administrator must therefore decide who can modify data, data structure and tables and what types of changes they can make to stop unauthorized access to sensitive Personally Identifiable Information (PII).

SQL standard supports DAC and has the GRANT and REVOKE commands. Privileges signify permission to access a particular kind of SQL statement or other users' objects. The GRANT command can be used to afford privileges to users and the REVOKE command can be used to take privileges away from users. To ensure database security, administrators should only grant privileges to users that expressly require them to carry out tasks involved in their job. This minimises the risk of administrators viewing, modifying or deleting data unrelated to their job role. For instance, non-DBA-privileged users need not be granted ANY privileges such as DROP ANY TABLE or CREATE ANY TABLE or be permitted to perform actions that create, drop or modify database objects. Granting privileges to roles (i.e., a named group of privileges) rather than individual users facilitates account administration and the management of privileges. We suggest separating roles/privileges for general library staff and library database administrators.

The network and its traffic need safeguarding from unauthorised access or modification and can be kept secure by using Secure Sockets Layer (SSL) when administering the listener (the listener receives incoming requests from clients to connect to the database server and manages this traffic). SSL offers authentication, encryption and data integrity - external or global users can authenticate to a database using SSL.

GDPR demands that data be managed securely to safeguard against unauthorized access, loss, processing, damage or destruction (*Data Protection Act* 2018). Library users should be aware of their right to have their personal data removed if they no longer need the service. Since the library's principal activity is to provide services to students and staff, their personal information is under the library's control. Administrators should determine which user data can be accessed in their current system activities and which data contains personal information. Article 4 of GDPR describes personal data as 'any information relating to an identified or identifiable natural person' (*Data Protection Act* 2018). This includes basic identity information- i.e., name (stored in the system's MEMBERSHIP table) and online identifiers like email addresses (stored in multiple table entities). Administrators should periodically review the sensitive data held by the database and remove or anonymise it when no longer needed.

Bibliography

- Data Protection Act 2018, c. 12 Available at http://www.legislation.gov.uk/ukpga/2018/12/contents/enacted (Accessed: 12 December 2020)