CHRIS HUI
DAVID NITU
EVELINA SARGSYAN
GEORGE SMAU

DATABASE SYSTEMS – ECS740P
COURSEWORK 2

17th DECEMBER 2021

Table of Contents

NOTE FROM GROUP 5:

Oracle SQL Live was unable to process the entirety of our code at once, likely due to its length.

Therefore, the code had been broken down into 3 text documents that should be ran in the following order:

First: Run_first_-_Create_tables_functions_procedures_triggers_and_views.txt
Second: Run_second_-_Test_data.txt
Third: Run_third_-_Test_statements.txt

1. INTRODUCTION

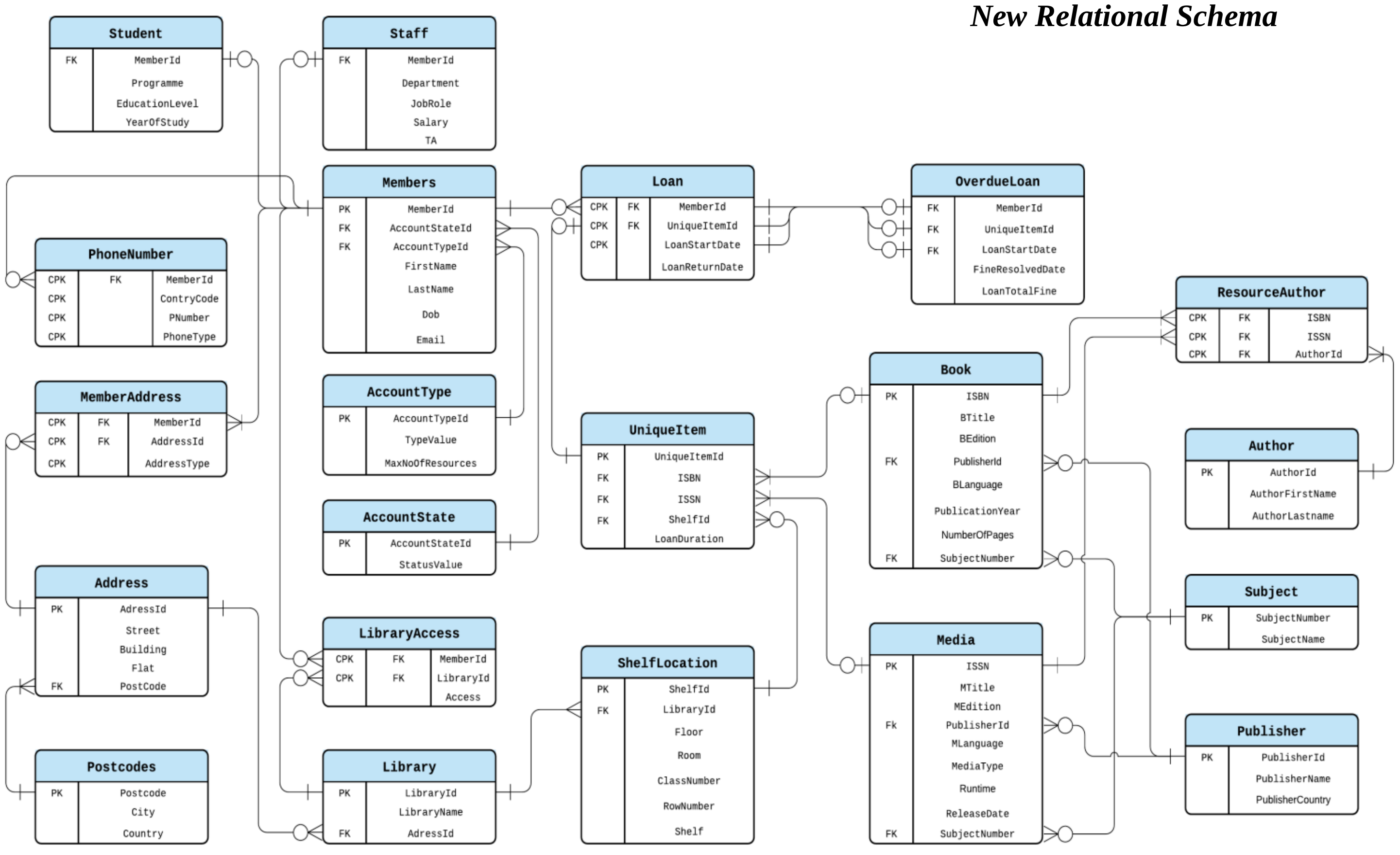
This report aims to outline the process of implementing the previously designed relational database using Oracle. This document will contain a breakdown of the following:

- The relational schema, along with any changes that have been made to it in relation to the previous iteration of the design.
- Table creation statements content explained.
- Test data content explained.
- Functions, views, queries and triggers along with their outputs explained.
- Functionality.
- Implementation, security and other concerns (AWS).
- Conclusion.

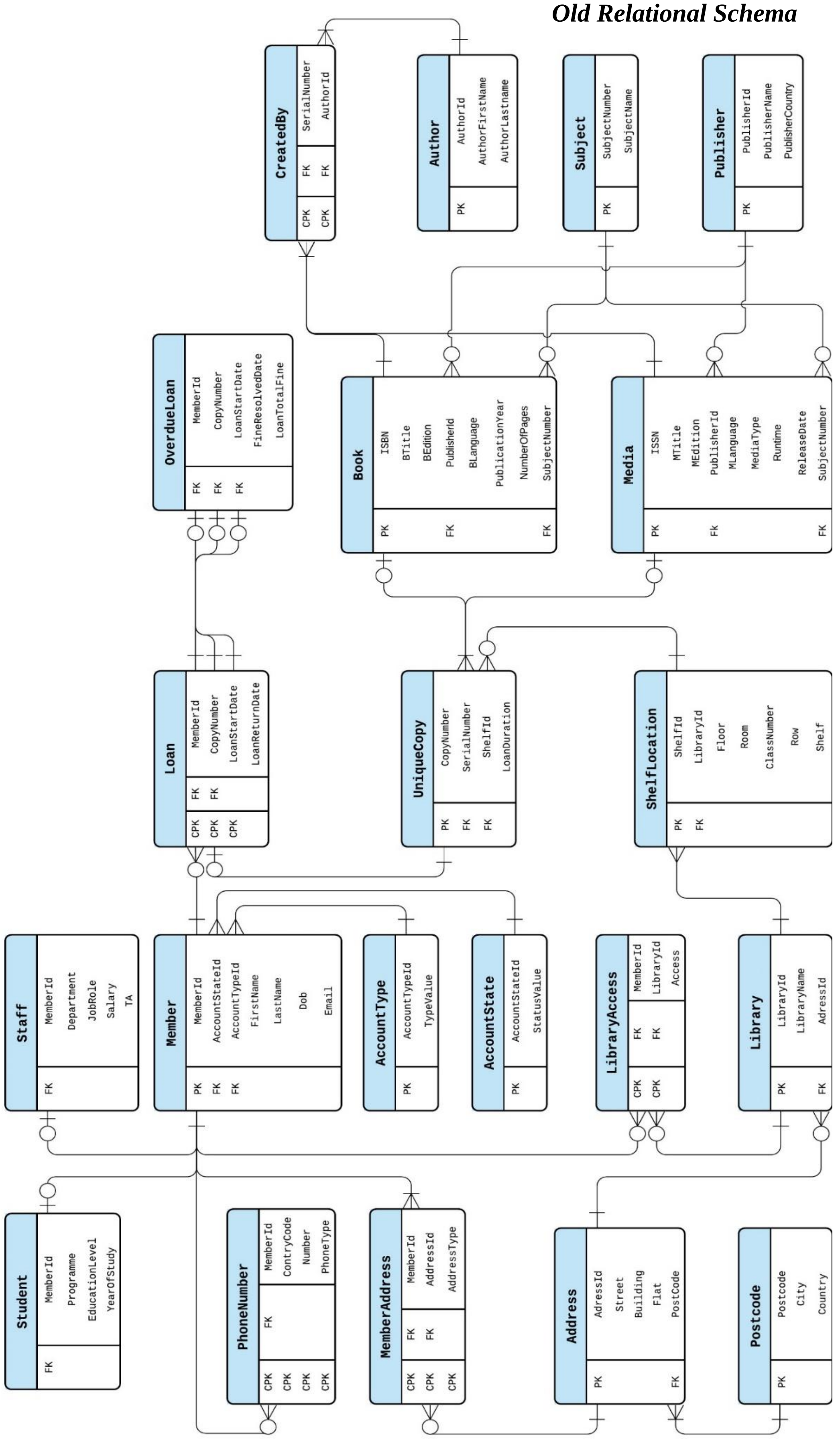
2. RELATIONAL SCHEMA

The relational schema design that has undergone some minor changes from its last iteration. These changes mostly involve renaming terms in order to avoid confusion both when implementing the database and at administration/user level. Modified entities are as follows:
Loan(*MemberId*, *UniqueItemId*, LoanStartDate, LoanReturnDate)
OverdueLoan(*MemberId*, *UniqueItemId*, *LoanStartDate*, FineResolvedDate, LoanTotalFine)
UniqueItem(UniqueItemId, ISBN, ISSN, ShelfId, LoanDuration)
ResourceAuthor(*ISBN*, *ISSN*, *AuthorId*)

New Relational Schema



Old Relational Schema



2. TABLE CREATION

3.1 Account type table

The AccountTypeId in this table has been given the primary key constraint. This means that the value within this column must be unique and not null. Furthermore, it is an integer sequentially and automatically generated “number generated always as identity”

The TypeValue has been set to Varchar2, denoting a string. Here we have the different types of members of the library, each being able to borrow a different number of resources at any given time as denoted by the max number of resources attribute “MaxNoOfResources”.

```
create table AccountType (
   AccountTypeId                    number generated always as identity primary key,
   TypeValue                        varchar2(20), -- Staff / Student / TA
   MaxNoOfResources                 number
);
```

3.2 Account state table

The AccountStateId in this table has been set to number and given the constraint of primary key, which as aforementioned makes sure it is unique and not null. Furthermore, it is an integer sequentially and automatically generated “number generated always as identity”

In similar fashion to the TypeValue in the Account type table, the StateValue has been set to Varchar2 in order to denote the different account states, in this case we have active, suspended (due to overdue fines) and closed (no longer a member).

```
create table AccountState (
   AccountStateId                   number generated always as identity primary key,
   StateValue                       varchar2(20) -- Active / Suspended / Closed
);
```

3.3 Members table

The MemberId in this table has been set to number and given the constraint of primary key, which as aforementioned makes sure it is unique and not null.

The foreign key fk_MemberType references the AccountTypeId primary key in the AccountType table and the fk_MemberState foreign key references the AccountStateId primary key from the AccountState table.

A further constraint has been added to the Email field to make sure the string that is input into it contains and ‘@ ’sign and a ‘. ’sign.

```
create table Members (
   MemberId                         number,
   FirstName                        varchar2(40) not null,
   LastName                         varchar2(40) not null,
   Dob                              date,
   Email                            varchar2(40) not null,
   AccountStateId                   number not null,
   AccountTypeId                    number not null,

    constraint pk_Member primary key (MemberId),
   constraint fk_MemberType foreign key (AccountTypeId)
      references AccountType (AccountTypeId),
    constraint fk_MemberState foreign key (AccountStateId)
      references AccountState (AccountStateId),
      check (Email like '%@%.%')
);
```

3.4 Staff table

The MemberId attribute in this table has been given both a primary and foreign key constraint.
The foreign key has been given an on delete cascade constraint. This means that when the attribute value is deleted in the parent table, the entry in this table will also be deleted.

```
create table Staff (
   MemberId                          number,
   Department                        varchar2(20),
   JobRole                           varchar2(20),
   salary                                    number,
   TA                                        char(1), -- Boolean, input: (Y/N)

   constraint pk_StaffId primary key (MemberId),
   constraint fk_Staff foreign key (MemberId)
      references Members (MemberId) on delete cascade
);
```

3.5 Student table

The MemberId attribute in this table has been given both a primary and foreign key constraint.
The foreign key has been given an on delete cascade constraint. This means that when the attribute value is deleted in the parent table, the entry in this table will also be deleted.

```
create table Student (
   MemberId                          number,
   Programme                         varchar2(100),
   EducationLevel                    varchar2(10),
   YearOfStudy                       varchar2(1),

   constraint pk_StudentId primary key (MemberId),
   constraint fk_Student foreign key (MemberId)
      references Members (MemberId) on delete cascade
);
```

3.6 Phone number table

This table's primary key constraint is composed of more than one attribute, making it a composite primary key (MemberId, CountryCode, PNumber, PhoneType).
The MemberId attribute in this table has been given both a primary and foreign key constraint.
The foreign key has been given an on delete cascade constraint. This means that when the attribute value is deleted in the parent table, the entry in this table will also be deleted.

```
create table PhoneNumber (
   MemberId                          number not null,
   CountryCode                       varchar2(10) not null,
   PNumber                                   varchar2(20) not null,
   PhoneType                         varchar2(30) not null,

      constraint pk_PhoneNumber primary key (MemberId, CountryCode, PNumber, PhoneType),
      constraint fk_PhoneNumber foreign key (MemberId)
            references Members (MemberId) on delete cascade
);
```

3.7 Postcodes table

The Postcode column in this table has been set to varchar2 (string) with a maximum character length of 10 to account for different postcodes from different countries, and given the constraint of primary key, which as aforementioned makes sure it is unique and not null.

```
create table PostCodes (
   PostCode                                varchar2(10),
   City                                    varchar2(30) not null,
   Country                                 varchar2(30) not null,

   constraint pk_PostCodes primary key (PostCode)
);
```

3.8 Address table

The primary key, addressId, in the Address table will be an incrementally generated number. The foreign key constraint fk_PostCode references the Postcode primary key in the Postcodes.

```
create table Address (
   AddressId                               number generated always as identity primary key,
   Street                                  varchar2(30) not null,
   Building                                varchar2(30) ,
   Flat                                    varchar2(10) ,
   PostCode                                varchar2(10) not null,

   constraint fk_PostCode foreign key (PostCode)
      references PostCodes (PostCode)
);
```

3.9 Member Address table

The primary key constraint within the MemberAddress table is a composite primary key constraint consisting of MemberId, AddressId, and AddressType.
The foreign key constraints references MemberId and AddressId of the Member and Address table respectively.
The on delete cascade indicates that the MemberId and AdddressId rows within this child table will be deleted when corresponding rows are deleted in the parent tables.

```
create table MemberAddress (
   MemberId                                number,
   AddressId                               number,
   AddressType                             varchar2(30),
   --pk/fk
   constraint pk_MemberAddress primary key (MemberId, AddressId, AddressType),
   constraint fk_MemberAddress1 foreign key (MemberId)
      references Members (MemberId) on delete cascade,
   constraint fk_MemberAddress2 foreign key (AddressId)
      references Address (AddressId) on delete cascade
);
```

3.10 Library table

The primary key of the library table will be an Id number that is generated incrementally for each library data entry. The foreign key constraint references AddressId of the Address table.

```
create table Lbry (
   LibraryId                               number generated always as identity primary key,
   LibraryName                             varchar2(50) not null,
   AddressId                               number,
   --fk
   constraint fk_Library foreign key (AddressId)
      references Address (AddressId)
);
```

3.11 Library Access table

The primary key constraint of the Library Access table will be a composite primary key of MemberId and Library Id. The foreign key constraints reference the LibraryId and MemberId of Library and Member tables respectively.

```
create table Lbry_Access (
   MemberId                    number not null,
   LibraryId                   number not null,
   AccessId                    varchar2 (50),
   --pk/fk
   constraint pk_Lbry_Access primary key (MemberId, LibraryId),
   constraint fk_Lbry_Access1 foreign key (LibraryId)
      references LBRY (LibraryId) on delete cascade,
   constraint fk_Lbry_Access2 foreign key (MemberId)
      references Members (MemberId) on delete cascade
);
```

3.12 Shelf Location table

The primary key constraint of the Shelf Location table will be the ShelfId, with foreign key constraint referencing the Library table. The Unique Constraint unicity indicates that the combination of fields (LibraryId, Floor, Room, RowNumber, Shelf) must uniquely define a record.

```
create table Shelf_location (
   ShelfId                     varchar2 (50),
   LibraryId                    number,
   Floor                       number,
   Room                        varchar2 (10),
   ClassNumber                 varchar2 (20),
   RowNumber                   varchar2 (20),
   Shelf                       varchar2 (10),
   --pf/fk
   constraint pk_Shelf primary key (ShelfId),
   constraint fk_Shelf foreign key (LibraryId)
      references Lbry (LibraryId),
   constraint unicity Unique (LibraryId, Floor, Room, RowNumber, Shelf)
);
```

3.13 Author table

In the Author table, the primary key AuthorId is generated. Here the check constraint ensures that either AuthorFirstname or AuthorLastname is not null.

```
create table Author (
   AuthorId                    number generated always as identity primary key,
   AuthorFirstname    varchar2(50),
   AuthorLastname        varchar2(50),

   check (((AuthorFirstName) is not null) or ((AuthorLastName) is not null))
);
```

3.14 Publisher table

PublisherId is a generated primary key for this table. PublisherName has a constraint not null.

```
create table Publisher(
   PublisherId                 number generated always as identity primary key,
   PublisherName               varchar2(100) not null,
   PublisherCountry            varchar(57)
);
```

3.15 Subject table

Here the primary key SubjectNumber is also generated and SubjectName has a constraint not null.

```
create table Subject(
   SubjectNumber      number generated always as identity primary key,
   SubjectName        varchar2(50) not null
);
```

3.16 Book table

For Book table ISBN is the primary key and given the appropriate constraint, which makes sure it is not null. BTitle and BLanguage have not null constraint.

The foreign keys PublisherId and SubjectNumber reference Publisher and Subject tables primary keys respectively.
create table Book (

```
    ISBN             varchar(13),
   BTitle                              varchar2(150) not null,
   BEdition                               varchar2(40),
   PublisherId                            number,
   BLanguage                              varchar2(20) not null,
   PublicationYear     date,
   NumberOfPages          number,
   SubjectNumber          number,
   --pk/fk
   constraint pk_Book primary key (ISBN),
   constraint fk_Book_PublisherId foreign key (PublisherId) references Publisher,
   constraint fk_Book_SubjectNumber foreign key (SubjectNumber) references Subject
);
```

3.17 Media table

For Media table ISSN is the primary key and given the appropriate constraint, which makes sure it is not null. MTitle and MLanguage have not null constraint.

The foreign keys PublisherId and SubjectNumber reference Publisher and Subject tables primary keys respectively.

```
create table Media (
   ISSN          varchar2(8),
   MTitle        varchar2(100) not null,
   MEdition        varchar2(40),
   PublisherId                         number,
   MLanguage          varchar2(20) not null,
   MediaType                           varchar2(7),
   Runtime                             varchar2 (16),
   ReleaseDate          date,
   SubjectNumber      number,
   --pk/fk
   constraint pk_Media                primary key (ISSN),
   constraint fk_Media_PublisherId foreign key (PublisherId) references Publisher,
   constraint fk_Media_SubjectNumber foreign key (SubjectNumber) references Subject
);
```

3.18 Resource Author table

Being an association table ResourceAuthor consist of foreign keys referencing primary keys of Book, Media and AuthorId referencing Author table primary has been given an on delete cascade constraint guaranteeing the entry in this table will also be deleted if the attribute value is deleted in the parent table.

```
create table ResourceAuthor(

   ISBN          varchar2(13),

   ISSN          varchar2(13),

   AuthorId       number,

   --pk/fk

   constraint primarykey unique (ISBN, ISSN, AuthorId),

   constraint fk_ResourceAuthor_AuthorId foreign key (AuthorId) references Author on delete cascade,
   constraint fk_ResourceAuthor_ISBN foreign key (ISBN)
       references Book on delete cascade,
   constraint fk_ResourceAuthor_ISSN foreign key (ISSN)
       references Media on delete cascade,
   check ((ISBN is null and ISSN is not null) or (ISSN is null and ISBN is not null))
);
```

3.19 Resource UniqueItem

The primary key UniqueItemIdentifier is generated providing uniqueness for each entry. Both LocationId and Loanduration have not null constraints. The foreign keys ISBN and ISSN with delete cascade referencing Book and Media respectively. Check constraint assures either ISBN or ISSN is not null.

```
create table UniqueItem(
    UniqueItemIdentifier    number generated always as identity primary key,
    ISBN                    varchar2(13),
    ISSN                    varchar2(8),
    LocationId              varchar2(10) not null,
    Loanduration            interval day to second not null,
    --fk
    constraint fk_Book_ISBN foreign key (ISBN)
        references Book (ISBN) on delete cascade,
    constraint fk_Media_ISSN foreign key (ISSN)
        references Media (ISSN) on delete cascade,
    check ((ISBN is null and ISSN is not null) or (ISSN is null and ISBN is not null))
);
```

3.20 Loan

This table has composite primary key MemberId, UniqueItemIdentifier, LoanStartDate and have primary key constraint, thus can’t be null. The foreign key fk_Loan_1 references the UniqueItemIdentifier primary key from UniqueItem table and fk_Loan_2 foreign key references the MemberId primary key from Members table.

```
create table Loan (
    MemberId                number,
    UniqueItemIdentifier    number,
    LoanStartDate           timestamp with local time zone default on null localtimestamp,
    LoanReturnDate          timestamp default null,
    -- pk/fk
    constraint pk_Loan primary key (MemberId, UniqueItemIdentifier, LoanStartDate),
    constraint fk_Loan_1 foreign key (UniqueItemIdentifier)
        references UniqueItem (UniqueItemIdentifier) on delete cascade,
    constraint fk_Loan_2 foreign key (MemberId)
        references Members (MemberId) on delete cascade
);
```

3.21 OverdueLoan

The constraint pk_OverdueLoan is a primary key consraint, which includes MemberId, UniqueItemIdentifier, LoanStartDate columns, and can't be null. The constraint fk_OverdueLoan_1 foreign key references the primary key from Loan table and has on delete cascade constraint.

```
create table OverdueLoan (
    MemberId              number,
    UniqueItemIdentifier    number,
    LoanStartDate          timestamp with local time zone default on null localtimestamp,
    FineResolvedDate       timestamp default null,
    LoanTotalFine          number default null,
    -- pk/fk
    constraint pk_OverdueLoan primary key (MemberId, UniqueItemIdentifier, LoanStartDate),
    constraint fk_OverdueLoan_1 foreign key (MemberId, UniqueItemIdentifier, LoanStartDate)
        references Loan (MemberId, UniqueItemIdentifier, LoanStartDate) on delete cascade
);
```

3. TEST DATA

4.1 Account type table

In this table, we insert the three types of Member account followed by the maximum number of resources they can borrow at an any given time. As the AccountTypeId is generated automatically in sequence, we do not need to write an insert into statement for it.

```
insert into AccountType (TypeValue, MaxNoOfResources) values ('Staff', 10);
insert into AccountType (TypeValue, MaxNoOfResources) values ('Student', 5);
insert into AccountType (TypeValue, MaxNoOfResources) values ('Teaching Assistant', 10);
```

4.2 Account state table

In this table we have the three types of Account state.
Users are kept in the system for history and once they are no longer a member of the library the account state is set closed "closed". As the AccountStateId is generated automatically in sequence, we do not need to write an insert into statement for it.

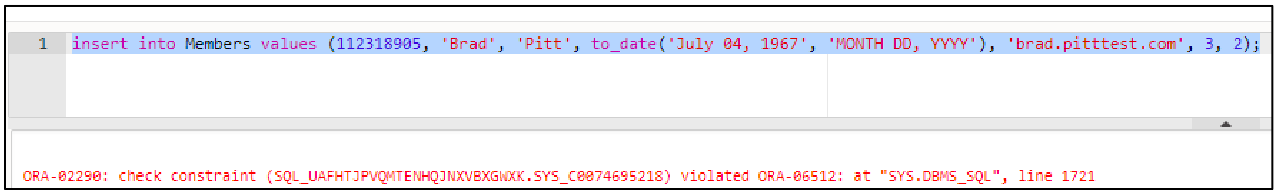
```
insert into AccountState (StateValue) values ('Active');
insert into AccountState (StateValue) values ('Suspended');
insert into AccountState (StateValue) values ('Closed');
```

4.3 Members table

In this table we have a variety that covers edge cases when it comes to testing out functionality. This comes in the form of a mix of account state and account. In this dataset, all emails are correct, however, in order to test the check constraint that we have set for it, we ran the following statement first:

```
insert into Members values (112318905, 'Brad', 'Pitt', to_date('July 04, 1967', 'MONTH DD, YYYY'), 'brad.pitttest.com', 3, 2);
```

Below is a screenshot of the output, confirming that we get a check constraint violation error:


```
insert into Members values (112318905, 'Brad', 'Pitt', to_date('July 04, 1967', 'MONTH DD, YYYY'), 'brad.pitttest.com', 3, 2);
```
```
ORA-02290: check constraint (SQL_UAFHTJPVQMTENHQJNXVBXGNXK.SYS_C0074695218) violated ORA-06512: at "SYS.DBMS_SQL", line 1721
```

The dates of birth in this dataset are also all within the correct parameters, however, we have created a trigger that validates that the date being input into the Dob column are after Jan 2nd, 1903 (currently oldest person alive) and older than 15 years old. This trigger will be demonstrated later in this document in the "Triggers" section

```
insert into Members values (210457221, 'Chris', 'Hui', to_date('December 23, 1991', 'MONTH DD, YYYY'), 'christopher.slhui@gmail.com', 1, 2);
insert into Members values (200901222, 'George', 'Smau', to_date('June 27, 1997', 'MONTH DD, YYYY'), 'alexandrudmau@gmail.com', 1, 3);
insert into Members values (160591543, 'David', 'Nitu', to_date('May 26, 1997', 'MONTH DD, YYYY'), 'nitu.david97@yahoo.com', 1, 2);
insert into Members values (120381301, 'Evelina', 'Sargsyan', to_date('November 21, 1995', 'MONTH DD, YYYY'), 'placeholder@gmail.com', 1, 1);
insert into Members values (397674890, 'Tom', 'Cruise', to_date('July 03, 1962', 'MONTH DD, YYYY'), 'tom.cruise@iscool.com', 1, 3);
insert into Members values (112318907, 'Ralph', 'Finnes', to_date('August 12, 1997', 'MONTH DD, YYYY'), 'ralph.finnes@test.com', 1, 2);
insert into Members values (112318905, 'Brad', 'Pitt', to_date('July 04, 1967', 'MONTH DD, YYYY'), 'brad.pitt@test.com', 3, 2);
```

4.4 Staff table

In this table, we have provided some extra details for the members that are staff or teaching assistants.

```
insert into Staff values (120381301, 'Life Sciences', 'Librarian', 29000, 'N');
insert into Staff values (200901222, 'Social Sciences', 'Librarian', 25000, 'Y');
insert into Staff values (397674890, 'Life Sciences', 'Librarian', 15000, 'Y');
```

4.5 Student table

In this table, we have provided some extra details for the members that are students.

```
insert into Student values (210457221, 'Computer Science', 'MSc', '1');
insert into Student values (200901222, 'Medicine', 'BSc','3');
insert into Student values (160591543, 'Human Rights', 'PhD', '5');
insert into Student values (397674890, 'Drama', 'BA','2');
insert into Student values (112318907, 'Construction Management', 'BSc','3');
insert into Student values (112318905, 'Mathematics', 'PhD', null);
```

4.6 Phone number table

In this table, we have used a combination of different country codes, numbers and home and work phone numbers.

```
insert into PhoneNumber values(210457221, '+44', '70 5927 8831', 'home');
insert into PhoneNumber values(210457221, '+44', '70 5927 8831', 'work');
insert into PhoneNumber values(200901222, '+39', '360 7042947', 'home');
insert into PhoneNumber values(200901222, '+420','582 391 334', 'home');
insert into PhoneNumber values(160591543, '+44', '79 8001 9124', 'work');
insert into PhoneNumber values(120381301, '+40', '254 308085', 'work');
insert into PhoneNumber values(112318907, '+44', '71 8451 9384', 'home');
insert into PhoneNumber values(112318905, '+40', '215 317095', 'work');
```

4.7 Postcodes table

In this table we stored the postcodes that we have used in our test data along with the associated country and City they belong to.

```
insert into PostCodes values('EC2A 3LX', 'London', 'United Kingdom');
insert into PostCodes values('E14 3NE', 'London', 'United Kingdom');
insert into PostCodes values('WC2A 3HP', 'London', 'United Kingdom');
insert into PostCodes values('SW7 2LQ', 'London', 'United Kingdom');
insert into PostCodes values('SW5 0PT', 'London', 'United Kingdom');
insert into PostCodes values('E14 5EZ', 'London', 'United Kingdom');
insert into PostCodes values('SW3 2BX', 'London', 'United Kingdom');
insert into PostCodes values('SW12 9BN', 'London', 'United Kingdom');
```

4.8 Address table

In this table we have the addresses that we have used for our members. As the AddressId is generated automatically in sequence, we do not need to write an insert into statement for it.

```
insert into Address (Street, Building, Flat, Postcode) values('32 Rivington St', 'Alto', '16', 'EC2A 3LX');
insert into Address (Street, Building, Flat, Postcode) values('3 Glengall Grove', 'Redhouse', '67', 'E14 3NE');
insert into Address (Street, Building, Flat, Postcode) values('2 Gate St', 'Blithehale Court', '112', 'WC2A 3HP');
insert into Address (Street, Building, Flat, Postcode) values('41 Thurloe St', 'Canada Gardens', '54', 'SW7 2LQ');
insert into Address (Street, Building, Flat, Postcode) values('20-22 Hogarth Rd', 'Goliath Bank', null, 'SW5 0PT');
insert into Address (Street, Building, Flat, Postcode) values('30 S Colonnade', 'Main Library', null, 'E14 5EZ');
insert into Address (Street, Building, Flat, Postcode) values('17-19 Egerton Terrace', 'Jeremy Bentham Library', null, 'SW3 2BX');
insert into Address (Street, Building, Flat, Postcode) values('134 Balham High Rd', 'Balham Library', null, 'SW12 9BN');
```

4.9 Member Address table

For the Member Address table, data is passed in to associate a particular member, an addressed, and the type of address e.g. home, work, term-time address

```
insert into MemberAddress values(200901222, 1, 'Home');
insert into MemberAddress values(200901222, 6, 'Work');
insert into MemberAddress values(397674890, 2, 'Home');
insert into MemberAddress values(397674890, 8, 'Work');
insert into MemberAddress values(120381301, 2, 'Home');
insert into MemberAddress values(120381301, 8, 'Work');
insert into MemberAddress values(160591543, 3, 'Home');
insert into MemberAddress values(112318907, 3, 'Home');
insert into MemberAddress values(210457221, 4, 'Term-time');
insert into MemberAddress values(112318905, 5, 'Work');
```

4.10 Library table

We have decided to implement 3 libraries along with a library access table to add functionality. As the LibraryId is generated automatically in sequence, we do not need to write an insert into statement for it.

```
insert into Lbry (LibraryName, AddressId) values('Main Library', 6);
insert into Lbry (LibraryName, AddressId) values('Jeremy Bentham Library', 7);
insert into Lbry (LibraryName, AddressId) values('Balham Library', 8);
```

4.11 Library Access table

This table has been populated by executing the getLibraryAccess procedure, which will be explained in the Procedure section.

```
execute grantLibraryAccess(210457221, 1);
execute grantLibraryAccess(210457221, 2);
execute grantLibraryAccess(210457221, 3);
execute grantLibraryAccess(200901222, 1);
execute grantLibraryAccess(200901222, 2);
execute grantLibraryAccess(200901222, 3);
execute grantLibraryAccess(160591543, 1);
execute grantLibraryAccess(160591543, 2);
execute grantLibraryAccess(160591543, 3);
execute grantLibraryAccess(120381301, 1);
execute grantLibraryAccess(120381301, 2);
execute grantLibraryAccess(120381301, 3);
execute grantLibraryAccess(397674890, 1);
execute grantLibraryAccess(397674890, 2);
execute grantLibraryAccess(112318907, 3);
```

4.12 Shelf Location table

Data in the shelf location table is populated with specific shelf locations in the library, relating to a specific class e.g. computer science, history, mathematics.

```
insert into Shelf_location values('JER72224', 2, 0, 'G.3', 'Mathematics', 'MAT1', 'AJ-AL');
insert into Shelf_location values('BAL82011', 3, 0, 'G.1', 'Anatomy', 'ANAT2', 'ER-ES');
insert into Shelf_location values('BAL65453', 3, 1, 'F1.5', 'Social sciences', 'SOC', 'AA-AB');
insert into Shelf_location values('MAI73772', 1, 2, 'F2.3', 'English Literature', 'ENLIT', 'TH-TI');
insert into Shelf_location values('MAI50994', 1, 3, 'F3.5', 'Self Development', 'SLFDEV', 'PR-PS');
insert into Shelf_location values('JER71099', 2, 1, 'F1.5', 'Computer Science', 'COMPSCI', 'PY-PZ');
insert into Shelf_location values('MAI62866', 1, 2, 'F2.3', 'French Language', 'FRLNG', 'VO-VP');
insert into Shelf_location values('JER73334', 2, 1, 'F1.5', 'Computer Science', 'COMPSCI', 'CL-CO');
insert into Shelf_location values('JER73534', 2, 1, 'F1.5', 'Computer Science', 'COMPSCI', 'AG-AI');
insert into Shelf_location values('JER68754', 2, 1, 'F1.5', 'Computer Science', 'COMPSCI', 'EX-FA');
insert into Shelf_location values('JER68154', 2, 1, 'F1.5', 'Computer Science', 'COMPSCI', 'IN-IM');
insert into Shelf_location values('JER85154', 2, 1, 'F1.5', 'Computer Science', 'COMPSCI', 'PR-PU');
insert into Shelf_location values('BAL81111', 3, 0, 'G.1.2', 'Sciences', 'BIO', 'PE-PH');
insert into Shelf_location values('BAL82121', 3, 0, 'G.1.2', 'Sciences', 'BIO', 'LI-LO');
insert into Shelf_location values('BAL63799', 3, 4, 'F4.1', 'Biology', 'BIO1', 'BL-BM');
insert into Shelf_location values('MAI36861', 1, 4, 'F4.5', 'History', 'HIST5', 'WA-WE');
insert into Shelf_location values('JER60382', 2, 4, 'F4.3', 'Mathematics', 'MAT3', 'IN-IO');
insert into Shelf_location values('BAL84580', 3, 4, 'F4.1', 'Biology', 'BIO1', 'AS-AT');
insert into Shelf_location values('MAI97814', 1, 4, 'F4.2', 'Italian Language', 'ITLLANG','IT-IU');
insert into Shelf_location values('BAL64899', 3, 4, 'F4.1', 'Biology', 'BIO1', 'TH-TI');
insert into Shelf_location values('BAL66999', 3, 4, 'F4.1', 'Biology', 'BIO1', 'SE-SH');
insert into Shelf_location values('BAL69399', 3, 4, 'F4.1', 'Biology', 'BIO1', 'DA-DE');
```

4.13 Author table

Data in the author table represent books or media authors.

```
insert into Author(AuthorFirstname, AuthorLastname) values ('Alexey', 'Gorodentsev');
insert into Author(AuthorFirstname, AuthorLastname) values ('Elaine', 'Nicpon Marieb');
insert into Author(AuthorFirstname, AuthorLastname) values ('Judith', 'Bell');
insert into Author(AuthorFirstname, AuthorLastname) values ('F. Scott', 'Fitzgerald');
insert into Author(AuthorFirstname, AuthorLastname) values ('John', 'Medina');
insert into Author(AuthorFirstname, AuthorLastname) values ('Charles', 'Severance');
insert into Author(AuthorFirstname, AuthorLastname) values ('Gaël', 'Crépieux');
insert into Author(AuthorFirstname, AuthorLastname) values ('Caroline', 'Spérandio');
insert into Author(AuthorFirstname, AuthorLastname) values ('David', 'Attenborough');
insert into Author(AuthorFirstname, AuthorLastname) values ('Tony', 'Robinson');
insert into Author(AuthorFirstname, AuthorLastname) values ('Edward', 'Burger');
insert into Author(AuthorFirstname, AuthorLastname) values ('John', 'Campton');
insert into Author(AuthorFirstname, AuthorLastname) values ('Lingo Mastery', null);
insert into Author(AuthorFirstname, AuthorLastname) values ('Robert C.', 'Martin');
insert into Author(AuthorFirstname, AuthorLastname) values ('Micah', 'Martin');
insert into Author(AuthorFirstname, AuthorLastname) values ('James W.', 'Newkirk');
insert into Author(AuthorFirstname, AuthorLastname) values ('David', 'Clason');
insert into Author(AuthorFirstname, AuthorLastname) values ('Wilfried', ' Lemahieu');
insert into Author(AuthorFirstname, AuthorLastname) values ('Seppe vanden', 'Broucke');
```

4.15 Publisher table

Data in this table contains information relating to publishers: publisher name and publisher country.

```
insert into Publisher(PublisherName, PublisherCountry) values ('Springer', 'USA');
insert into Publisher(PublisherName, PublisherCountry) values ('Benjamin-Cummings Pub Co', 'USA');
insert into Publisher(PublisherName, PublisherCountry) values ('Open University Press', 'UK');
insert into Publisher(PublisherName, PublisherCountry) values ('Independently published', 'USA');
insert into Publisher(PublisherName, PublisherCountry) values ('CreateSpace Independent Publishing Platform', 'USA');
insert into Publisher(PublisherName, PublisherCountry) values ('The Best Media Publisher', 'USA');
insert into Publisher(PublisherName, PublisherCountry) values ('BBC Studio', 'UK');
insert into Publisher(PublisherName, PublisherCountry) values ('Lingo Mastery', 'Italy');
insert into Publisher(PublisherName, PublisherCountry) values ('Pearson Education', 'UK');
insert into Publisher(PublisherName, PublisherCountry) values ('Addison-Wesley', 'USA');
insert into Publisher(PublisherName, PublisherCountry) values ('BBC Books', 'UK');
```

4.16 Subject table

Data in this table contains the various subjects that is associated with resources.

```
insert into Subject(SubjectName) values ('Mathematics');
insert into Subject(SubjectName) values ('Sciences');
insert into Subject(SubjectName) values ('Social Sciences');
insert into Subject(SubjectName) values ('English Literature');
insert into Subject(SubjectName) values ('French Language');
insert into Subject(SubjectName) values ('Italian Language');
insert into Subject(SubjectName) values ('Computer Science');
insert into Subject(SubjectName) values ('Anatomy');
insert into Subject(SubjectName) values ('Self Development');
insert into Subject(SubjectName) values ('Biology');
insert into Subject(SubjectName) values ('History');
```

4.17 Book table

Each entry in the Book table contains information relating to a specific book. Book title, book edition, book language etc. The values for publisher id and subject number relates to the Publisher table and Subject table mentioned above.

```
insert into Book values ('9783319508528', 'Algebra II: Textbook for Students of Mathematics',
'1st', 1, 'English', to_date('2017/01/03','yyyy/mm/dd'), 385, 1);
insert into Book values ('9780131934818', 'Essentials of Human Anatomy and Physiology',
null, 2, 'English', to_date('2005/02/06','yyyy/mm/dd'), 288, 8);
insert into Book values ('9780335215041', 'Doing Your Research Project: A Guide for First-Time Researchers in Education, Health and Social Science',
'4th', 3, 'English', to_date('2005/06/02','yyyy/mm/dd'), 288, 3 );
insert into Book values ('9798537197515', 'This Side of Paradise',
null, 4, 'English', to_date('2021/01/01','yyyy/mm/dd'), 212, 4 );
insert into Book values ('9798558182569', '12 Principles For Surviving and Thriving at Work, Home, and School',
null, 4, 'English', to_date('2020/03/03','yyyy/mm/dd'), 33, 9 );
insert into Book values ('9781530051120', 'Python for Everybody',
'1st', 5, 'English', to_date('2016/09/02','yyyy/mm/dd'), 247, 7);
insert into Book values ('9782278083404', 'Vocabulaire essentiel du francais',
'2nd', 8, 'French', to_date('2015/10/15','yyyy/mm/dd'), 239, 5);
insert into Book values ('9780132350884', 'Clean Code: A Handbook of Agile Software Craftsmanship',
null, 9, 'English', to_date('2008/08/01','yyyy/mm/dd'), 464, 7);
insert into Book values ('9780135974445', 'Agile Software Development, Principles, Patterns, and Practices',
null, 9, 'English', to_date('2002/08/25','yyyy/mm/dd'), 552, 7);
insert into Book values ('9780131857254', 'Agile Principles, Patterns, and Practices in C#',
null, 9, 'English', to_date('2006/07/20','yyyy/mm/dd'), 768, 7);
insert into Book values ('9780201709377', 'Extreme Programming in Practice',
null, 10, 'English', to_date('2006/07/20','yyyy/mm/dd'), 224, 7);
insert into Book values ('9781511654944', 'Introduction to Networking: How the Internet Works',
'1st', 5, 'English', to_date('2015/04/29','yyyy/mm/dd'), 112, 1);
insert into Book values ('9798773225324', 'Introduction to Algorithms Book',
'3rd', 4, 'English', to_date('2021/11/24','yyyy/mm/dd'), 37, 1);
insert into Book values ('9781107186125', 'Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data',
'1st', 3, 'English', to_date('2018/07/12','yyyy/mm/dd'), 808, 1);
insert into Book values ('9781785945298', 'A Perfect Planet',
'3rd',11, 'English', to_date('2020/10/20','yyyy/mm/dd'), 324, 2);
insert into Book values ('9780008477820', 'Living Planet',
'3rd',11, 'English', to_date('2020/10/20','yyyy/mm/dd'), 338, 2);
```

4.18 Media table

Each entry in the Media table contains information relating to a specific media item. Similar to the Book table, the values for publisher id and subject number relates to the Publisher table and Subject table.

```
insert into Media values('25698345', 'Blue Planet II ', '2nd', 7, 'English', 'DVD', '6hour 40min',
to_date('2017/04/15','yyyy/mm/dd'), 10);
insert into Media values('21459876', 'Walking Through History', '3rd', 7, 'English', 'DVD', '4hour 40min',
to_date('2015/01/01','yyyy/mm/dd'), 11);
insert into Media values('65987412', 'Introduction to Number Theory', '1st', 6, 'English', 'DVD', '2hour 51min',
to_date('2015/05/04','yyyy/mm/dd'), 1);
insert into Media values('78954632', 'AS Biology Revision', '1st', 6, 'English', 'CD', '2hour 51min',
to_date('2006/08/09','yyyy/mm/dd'), 10);
insert into Media values('15623586', 'Italian Short Stories for Beginners', '1st', 8, 'Italian', 'CD', '6hour 19min',
to_date('2006/04/03','yyyy/mm/dd'), 6);
insert into Media values('25698254', 'The Penguin King', '1st', 7, 'English', 'DVD', '5hour 40min',
to_date('2020/02/04','yyyy/mm/dd'), 10);
insert into Media values('25691234', 'Seven Worlds, One Planet', '2st', 7, 'English', 'DVD', '4hour 45min',
to_date('2019/04/15','yyyy/mm/dd'), 10);
insert into Media values('25699874', 'David Attenborough The Definitive Collection Natural History Museum Alive, Galapagos, Micro Monsters', '1st', 7,
'English', 'DVD', '7hour 30min',
to_date('2019/04/22','yyyy/mm/dd'), 10);
```

4.19 Resource Author table

Each entry in the Resource Author table relates a Book or Media item to the appropriate Author. It should be noted that the null values seen below indicates the entry is *not* of that resource type.

```
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9783319508528', null, 1);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9780131934818',  null, 2);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9780335215041',  null, 3);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9798537197515',  null, 4);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9798558182569',  null, 5);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9781530051120', null, 6);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9782278083404', null, 7);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9782278083404', null, 8);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values (null, '25698345', 9);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values (null,'21459876', 10);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values (null,'65987412', 11);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values (null,'78954632', 12);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values (null,'15623586', 13);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9780132350884', null, 14);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9780135974445', null, 14);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9780131857254', null, 14);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9780131857254', null, 15);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9780201709377', null, 14);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9780201709377', null, 16);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9781511654944', null, 6);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9798773225324', null, 17);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9781107186125', null, 18);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9781107186125', null, 19);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9781785945298', null, 9);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values ('9780008477820', null, 9);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values (null, '25698254',9);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values (null, '25691234',9);
insert into ResourceAuthor(ISBN, ISSN, AuthorId) values (null, '25699874',9);
```

4.20 Unique Item table

Each entry in the Unique Item table represents one unique item. Similar to the data entries for the Resource Author table, the null values in the population indicates the item is *not* of that resource type. The maximum loan duration is inserted as 14days / 2 days / 0 days (to be used within the library only).

```
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9783319508528', null, 'JER72224', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780131934818', null, 'BAL82011', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780335215041', null, 'BAL65453', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9798537197515', null, 'MAI73772', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9798558182569', null, 'MAI50994', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781530051120', null, 'JER71099', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9782278083404', null, 'MAI62866', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780132350884', null, 'JER73334', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780135974445', null, 'JER73534', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780131857254', null, 'JER73534', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780201709377', null, 'JER68754', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781511654944', null, 'JER68154', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9798773225324', null, 'JER68154', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781107186125', null, 'JER85154', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781785945298', null, 'BAL81111', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780008477820', null, 'BAL82121', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25698345',  'BAL63799', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'21459876',  'MAI36861', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'65987412',  'JER60382', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'78954632',  'BAL84580', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'15623586',  'MAI97814', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25698254',  'BAL64899', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25691234',  'BAL66999', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25699874',  'BAL69399', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9783319508528', null, 'JER72224', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780131934818', null, 'BAL82011', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780335215041', null, 'BAL65453', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9798537197515', null, 'MAI73772', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9798558182569', null, 'MAI50994', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781530051120', null, 'JER71099', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9782278083404', null, 'MAI62866', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780132350884', null, 'JER73334', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780135974445', null, 'JER73534', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780131857254', null, 'JER73534', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780201709377', null, 'JER68754', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781511654944', null, 'JER68154', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9798773225324', null, 'JER68154', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781107186125', null, 'JER85154', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781785945298', null, 'BAL81111', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780008477820', null, 'BAL82121', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25698345',  'BAL63799', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'21459876',  'MAI36861', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'65987412',  'JER60382', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'78954632',  'BAL84580', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'15623586',  'MAI97814', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25698254',  'BAL64899', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25691234',  'BAL66999', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25699874',  'BAL69399', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'78954632',  'BAL84580', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'15623586',  'MAI97814', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25698254',  'BAL64899', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25691234',  'BAL66999', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values (null,'25699874',  'BAL69399', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9783319508528', null, 'JER72224', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780131934818', null, 'BAL82011', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780335215041', null, 'BAL65453', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9798537197515', null, 'MAI73772', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9798558182569', null, 'MAI50994', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781530051120', null, 'JER71099', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9782278083404', null, 'MAI62866', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780132350884', null, 'JER73334', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780135974445', null, 'JER73534', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781511654944', null, 'JER68154', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9798773225324', null, 'JER68154', INTERVAL '2' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781107186125', null, 'JER85154', INTERVAL '0' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9781785945298', null, 'BAL81111', INTERVAL '14' DAY);
insert into UniqueItem (ISBN, ISSN, LocationId, Loanduration) values ('9780008477820', null, 'BAL82121', INTERVAL '14' DAY);
```

4.21 Loan table & 4.22 Overdue Loan table

Warning: In order to populate these tables, we will be using functions. The user should only use loanItem, returnItem and payFines procedures in order to avoid inserting erroneous data. This is because these procedures check for the integrity of the data inserted.

```
-- Insert old dates into loan and pay the fines so that overdue loan table is pupulated
insert into Loan  values (120381301, 1, TIMESTAMP '2021-11-14 11:14:00', null);
insert into Loan  values (120381301, 2, TIMESTAMP '2021-11-15 10:34:00', null);
execute payFines(120381301);

-- In order to have different payment dates, wait one minute before running the next sections.
-- The payments are recorded to the minute, but not to the second, so if two payments are made
-- in the same minute they will be grouped as one.
insert into Loan  values (120381301, 3, TIMESTAMP '2021-11-16 11:14:00', null);
insert into Loan  values (120381301, 4, TIMESTAMP '2021-11-17 10:34:00', null);
execute payFines(120381301);

-- In order to have different payment dates, wait one minute before running the next sections.
-- The payments are recorded to the minute, but not to the second, so if two payments are made
-- in the same minute they will be grouped as one
```

```
insert into Loan  values (160591543, 6, TIMESTAMP '2021-11-18 11:14:00', null);
insert into Loan  values (160591543, 7, TIMESTAMP '2021-11-19 10:34:00', null);
execute payFines(160591543);

-- In order to have different payment dates, wait one minute before running the next sections.
-- The payments are recorded to the minute, but not to the second, so if two payments are made
-- in the same minute they will be grouped as one
insert into Loan  values (160591543, 8, TIMESTAMP '2021-11-20 11:14:00', null);
insert into Loan  values (160591543, 9, TIMESTAMP '2021-11-21 10:34:00', null);
execute payFines(160591543);

-- Insert active overdue loans by inserting loans with an old start date and performing a daily update
insert into Loan  values (120381301, 11, TIMESTAMP '2021-11-22 11:14:00', null);
insert into Loan  values (120381301, 12, TIMESTAMP '2021-11-23 10:34:00', null);
insert into Loan  values (160591543, 13, TIMESTAMP '2021-11-24 11:14:00', null);
insert into Loan  values (160591543, 15, TIMESTAMP '2021-11-25 10:34:00', null);
execute dailyUpdate;

-- Generating returned loaned entries by loaning an item and imediately returning it
execute loanItem(1,210457221);
execute loanItem(2,210457221);
execute loanItem(3,210457221);
execute returnLoan(1);
execute returnLoan(2);
execute returnLoan(3);

-- insert active loans into loan table
execute loanItem(1,210457221);
execute loanItem(2,210457221);
execute loanItem(3,210457221);
execute loanItem(4,200901222);
execute loanItem(6,200901222);
execute loanItem(7,200901222);
```

5. FUNCTIONS

In this section, we will be going over the functions that we have implemented. For each function there will be a prerequisite line at the very beginning

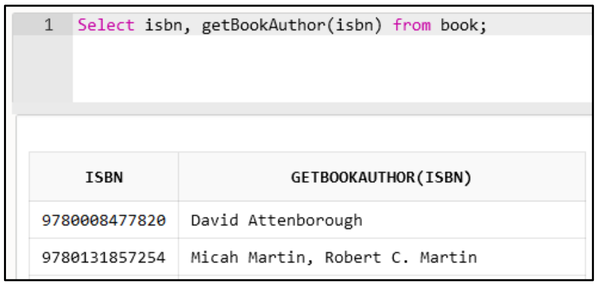
5.1 Get book author

The getBookAuthor function takes the input of Unique Resource Identifier (URI) ISBN, and returns the author of the queried book.

```
Create or replace function getBookAuthor
  (URI in varchar2)
  return varchar2
    is
  Authors varchar2(500);
Begin
  Select listagg(Author, ', ' ) within group (order by Author) into Authors from (Select Book.ISBN, (AuthorFirstname || ' ' || AuthorLastName)
Author from Book
    left join ResourceAuthor on ResourceAuthor.ISBN = Book.ISBN left join Author on ResourceAuthor.Authorid = Author.AuthorID) where
ISBN = URI group by ISBN order by ISBN;
    return Authors;
End;
```

Test statement:

Select isbn, getBookAuthor(isbn) from book;



```
Select isbn, getBookAuthor(isbn) from book;
```

ISBN	GETBOOKAUTHOR(ISBN)
9780008477820	David Attenborough
9780131857254	Micah Martin, Robert C. Martin

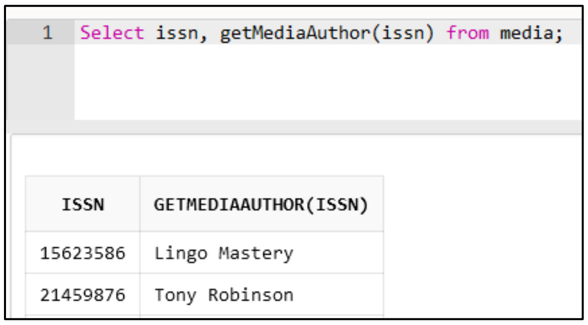
5.2 Get media author

The getMediaAuthor funciton takes the input of URI ISSN, and returns the author of the queried media resource.

```
Create or replace function getMediaAuthor
  (URI in varchar2)
  return varchar2
    is
  Authors varchar2(500);
Begin
  Select listagg(Author, ', ' ) within group (order by Author) into Authors from (Select Media.ISSN, (AuthorFirstname || ' ' || AuthorLastName)
Author from Media
    left join ResourceAuthor on ResourceAuthor.ISSN = Media.ISSN
    left join Author on ResourceAuthor.Authorid = Author.AuthorID)
    where ISSN = URI group by ISSN order by ISSN;
  Return Authors;
End;
/
```

Test statement:

Select issn, getMediaAuthor(issn) from media;



```
Select issn, getMediaAuthor(issn) from media;
```

ISSN	GETMEDIAAUTHOR(ISSN)
15623586	Lingo Mastery
21459876	Tony Robinson

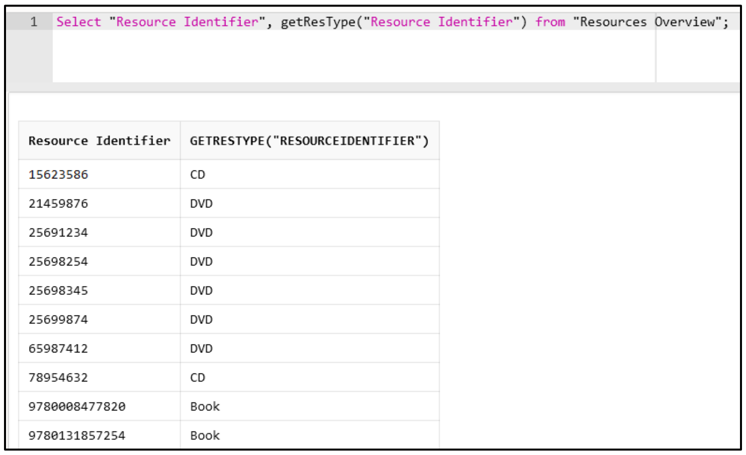
5.3 Get resource type

The getResType function takes the input of URI, and returns the appropriate type (e.g. ‘Book’, ‘CD’, ‘DVD’) of that resource.

```
Create or replace function getResType
  (URI in varchar2)
  return varchar2
    is
  ResType varchar2(50);
Begin
  Select nvl(mediatype, 'Book') into ResType from media full outer join book on media.issn = book.isbn where URI = NVL(media.issn,
book.isbn);
    return ResType;
End;
/
```

Test statement:

Select "Resource Identifier", getResType("Resource Identifier") from "Resources Overview";



```
Select "Resource Identifier", getResType("Resource Identifier") from "Resources Overview";
```

Resource Identifier	GETRESTYPE("RESOURCEIDENTIFIER")
15623586	CD
21459876	DVD
25691234	DVD
25698254	DVD
25698345	DVD
25699874	DVD
65987412	DVD
78954632	CD
9780008477820	Book
9780131857254	Book

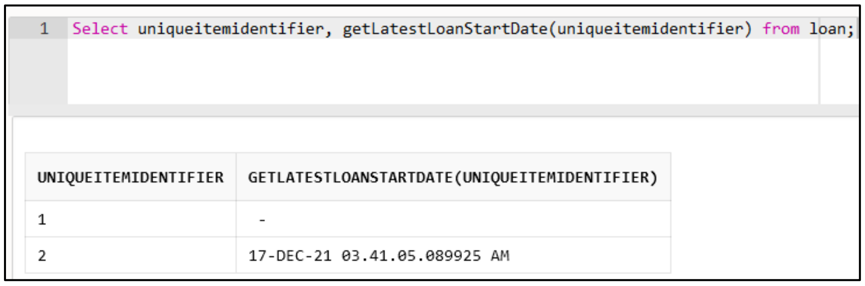
5.4 Get latest loan start date

The getLatestLoanStartDate function returns the start date of an active loan for a certain unique item. If the item is not currently loaned, it returns null.

```
Create or replace function getLatestLoanStartDate
  (UII in number)
  return timestamp
    is
  noOfRecords number;
  StartDate timestamp;
Begin
  Select count(*) into noOfRecords from Loan where UniqueItemIdentifier = UII and LoanReturnDate is null;
  if noOfRecords > 0 then
    Select LoanStartDate into StartDate from Loan where UniqueItemIdentifier = UII and LoanReturnDate is null;
  else StartDate := null;
  end if;
  return StartDate;
End;
/
```

Test statement:

Select uniqueitemidentifier, getLatestLoanStartDate(uniqueitemidentifier) from loan;



```
Select uniqueitemidentifier, getLatestLoanStartDate(uniqueitemidentifier) from loan;
```

UNIQUEITEMIDENTIFIER	GETLATESTLOANSTARTDATE(UNIQUEITEMIDENTIFIER)
1	-
2	17-DEC-21 03.41.05.089925 AM

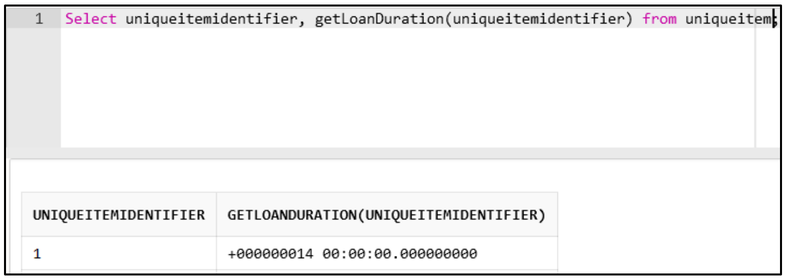
5.5 Get loan duration

The getLoanDuration function takes in the parameter Unique Item Identifier (UII), and returns the available duration of loan of the particular item, which can be 14 days, 2 days, or 0 days (e.g. to be used within the library).

```
Create or replace function getLoanDuration
  (UII in number)
  return interval day to second
    is
  loanDuration interval day to second;
Begin
     select uniqueItem.loanDuration into loanDuration from uniqueitem where uniqueitem.uniqueitemidentifier = UII;
    return loanDuration;
End;
/
```

Test statement:

Select uniqueitemidentifier, getLoanDuration(uniqueitemidentifier) from uniqueitem;



```
Select uniqueitemidentifier, getLoanDuration(uniqueitemidentifier) from uniqueitem;
```

UNIQUEITEMIDENTIFIER	GETLOANDURATION(UNIQUEITEMIDENTIFIER)
1	+000000014 00:00:00.000000000

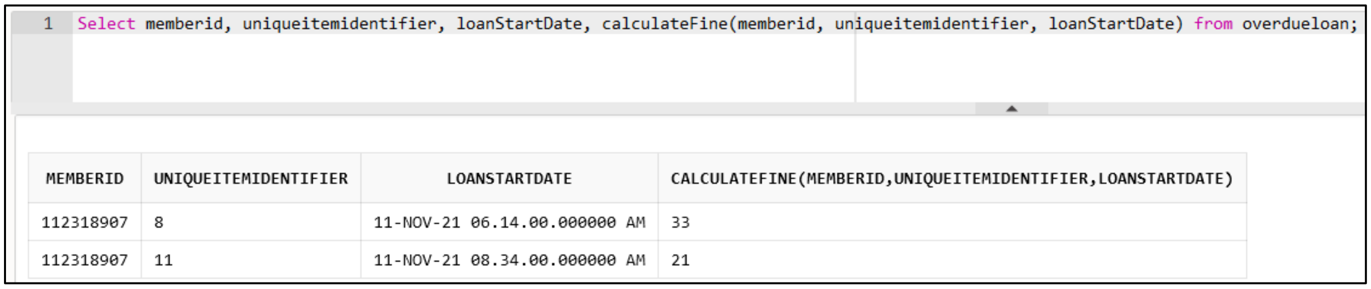
5.6 Calculate fine

The calculateFine function takes in the parameters Member ID, UII, UII's loan start time, and computes the total amount of fine owed by the library member on the particular item. A $1 fine per day is levied against the member for item loaned that is overdue.

```
Create or replace function calculateFine
  (membernum in number, UII in number, loansdate in timestamp)
  return number
    is
  fine number;
Begin
     select extract(day from diff) into fine
        from (select (current_timestamp - loanstartdate - getLoanDuration(UII)) diff
              from overdueloan where uniqueitemidentifier = UII and memberid = membernum and loanstartdate = loansdate and (current_timestamp
- loanstartdate) > getLoanDuration(UII));
    return fine;
End;
/
```

Test statement:

Select memberid, uniqueitemidentifier, loanStartDate, calculateFine(memberid, uniqueitemidentifier, loanStartDate) from overdueloan;



```
Select memberid, uniqueitemidentifier, loanStartDate, calculateFine(memberid, uniqueitemidentifier, loanStartDate) from overdueloan;
```

MEMBERID	UNIQUEITEMIDENTIFIER	LOANSTARTDATE	CALCULATEFINE(MEMBERID,UNIQUEITEMIDENTIFIER,LOANSTARTDATE)
112318907	8	11-NOV-21 06.14.00.000000 AM	33
112318907	11	11-NOV-21 08.34.00.000000 AM	21

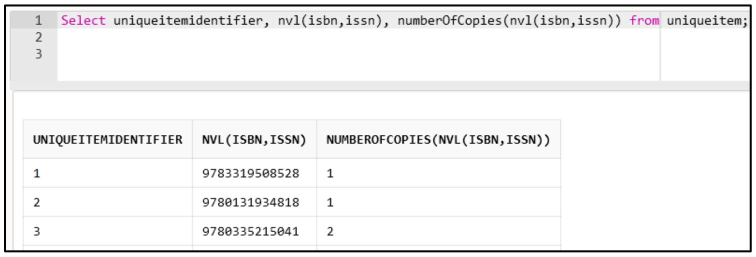
5.7 Get number of copies

The numberOfCopies function takes in the parameter URI, and returns the number of actively loaned copies of that resource.

```
Create or replace function numberOfCopies
  (URI in number)
  return number
    is
  copies number;
  loaned number;
Begin
      select count(*) into loaned from loan left join uniqueitem on loan.uniqueitemidentifier = uniqueitem.uniqueitemidentifier where (isbn = URI
or issn = URI) and loanreturndate is null;
      select count(*) into copies from uniqueitem where isbn = URI or issn = URI ;
    return copies-loaned;
End;
/
```

Test statement:

Select uniqueitemidentifier, nvl(isbn,issn), numberOfCopies(nvl(isbn,issn)) from uniqueitem;



```
Select uniqueitemidentifier, nvl(isbn,issn), numberOfCopies(nvl(isbn,issn)) from uniqueitem;
```

UNIQUEITEMIDENTIFIER	NVL(ISBN,ISSN)	NUMBEROFCOPIES(NVL(ISBN,ISSN))
1	9783319508528	1
2	9780131934818	1
3	9780335215041	2

5.8 Check item availability

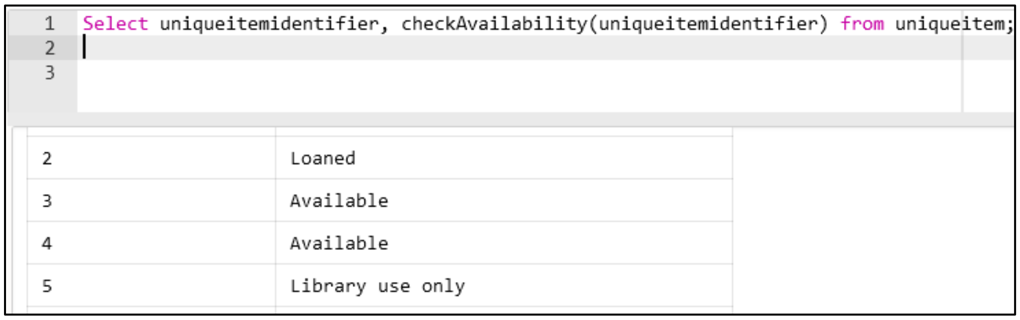
The checkAvailability function takes in the parameter UII, and return the availability status of the item. The function returns values of ‘Available’, ‘Loaned’ (indicating that the item is not available to loan), or ‘Library use only’.

```
Create or replace function checkAvailability
  (UII in number)
  return varchar2
    is
  status number;
  statusMessage varchar2(10);
Begin
    select count(*) into status from loan where UII = loan.uniqueitemidentifier and loanreturndate is null;

    if (getLoanDuration(UII) = INTERVAL '0' DAY) then
        return 'Library use only';
    elsif status = 1 then
         return 'Loaned';
    elsif status = 0 then
        return 'Available';
    end if;
End;
/
```

Test statement:

Select uniqueitemidentifier, checkAvailability(uniqueitemidentifier) from uniqueitem;



```
Select uniqueitemidentifier, checkAvailability(uniqueitemidentifier) from uniqueitem;
```

2	Loaned
3	Available
4	Available
5	Library use only

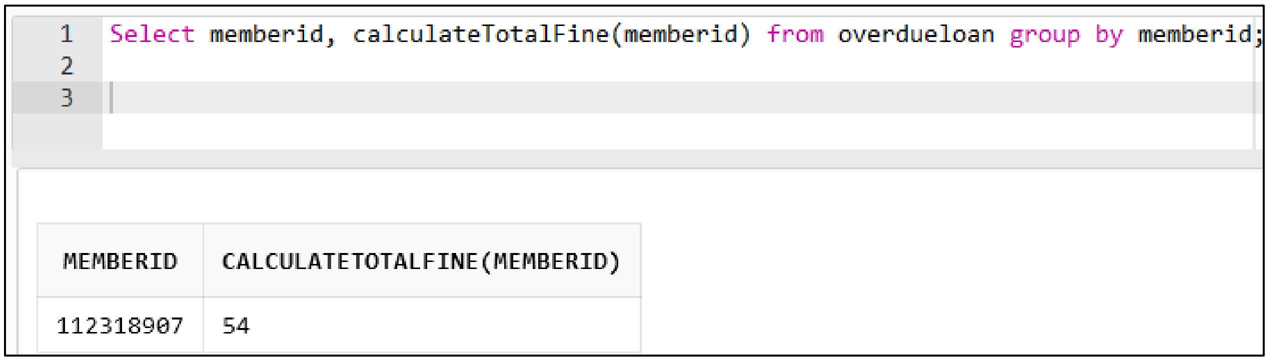
5.9 Calculate total fine

The calculateTotalFine function takes in the parameter Member Id, and calculates the total amount of fines owed by that particular member.

```
Create or replace function calculateTotalFine
    (membernum in number)
     return number
is
     currentTotalFines number;
Begin
     select sum (calculateFine(memberid, uniqueitemidentifier, loanstartdate)) into currentTotalFines from overdueloan where memberid =
     membernum and fineresolveddate is null group by memberid;
     return currentTotalFines;
End;
/
```

Test statement:

Select memberid, calculateTotalFine(memberid) from overdueloan group by memberid;



```
Select memberid, calculateTotalFine(memberid) from overdueloan group by memberid;
```

MEMBERID	CALCULATETOTALFINE(MEMBERID)
112318907	54

6. PROCEDURES

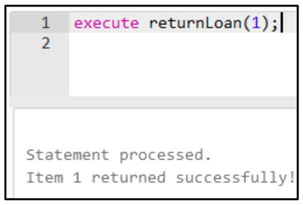
6.1 Return Loan

The returnLoan procedure takes in the parameter UII, and updates the loan's return date based on the current timestamp. This procedure would be called when an item is to be returned. If the item is not loaned, the function will return 'This item is not currently loaned!'.

```
Create or replace procedure returnLoan
 (UII number)
 is
 Startdate timestamp;
  Begin
  Startdate := getLatestLoanStartDate(UII);
  If Startdate is not null then
     update Loan
     set    Loan.Loanreturndate = current_timestamp
       where  Loan.uniqueitemidentifier = UII and Loan.loanstartdate = Startdate;
            dbms_output.put_line('Item ' UII ' returned successfully!');
  Else dbms_output.put_line('This item is not currently loaned!');
  End if;
  End;
/
```

Test statement:

execute returnLoan(2);



```
execute returnLoan(1);
```

```
Statement processed.
Item 1 returned successfully!
```

6.2 Update Loans

The updateLoans procedure populates the Overdue Loans table with new overdue loans.

This procedure is run within the dailyUpdate procedure.

```
Create or replace procedure updateLoans
  (membernum in number)
    is
  constraintcheck number;
  cursor loans
    is
  select MemberId, UniqueItemIdentifier, LoanStartDate from loan where (current_timestamp - LoanStartDate) > getLoanDuration(UniqueItemIdentifier) and loanReturnDate is null and memberid = membernum;
  begin
    for eachrow in loans
    loop
           select count(*) into constraintcheck from overdueloan where MemberId = eachrow.MemberId and UniqueItemIdentifier = eachrow.UniqueItemIdentifier and LoanStartDate = eachrow.LoanStartDate;
           if constraintcheck = 0 then
        Insert into OverDueLoan (MemberId, UniqueItemIdentifier, LoanStartDate)
          values(eachrow.MemberId, eachrow.UniqueItemIdentifier, eachrow.LoanStartDate);
      end if;
    end loop;
  end;
/
```

6.3 Pay Fines

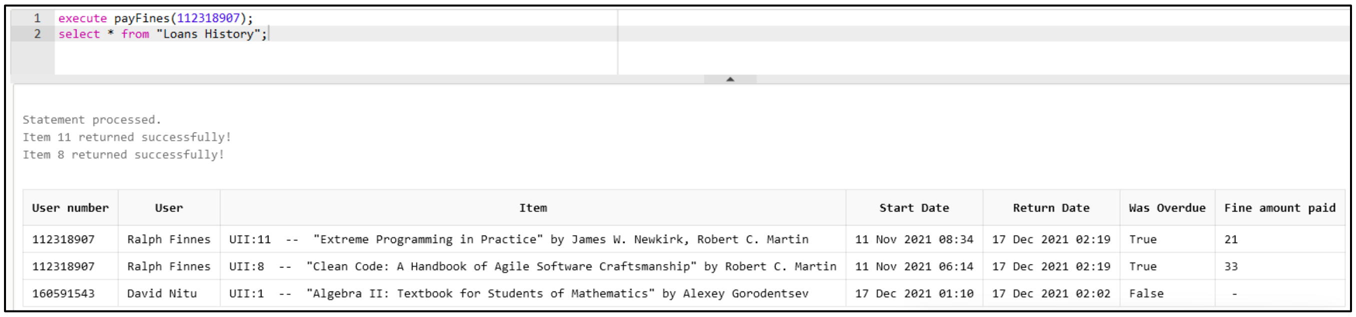
The payFines procedure takes in the parameter MemberId and resolves the fines incurred for overdue loans associated with the member. The procedure returns the overdue loans as well as resolving all outstanding fines.

```
Create or replace procedure payFines
   (membernum number)
      is
   fine number;
   memberstatus number;
   cursor loans
      is
   select
      uniqueitemidentifier, loanstartdate
   from
      overdueloan
   where memberid = membernum and fineresolveddate is null;
   begin
           updateLoans(membernum);
      for eachrow in loans
      loop
      fine := calculateFine(membernum, eachrow.uniqueitemidentifier, eachrow.loanstartdate);
      update OverDueLoan
      set   OverDueLoan.FineResolvedDate = current_timestamp, OverDueLoan.loantotalfine = fine
      where   OverDueLoan.memberid = memberNUM and OverDueLoan.uniqueitemidentifier = eachrow.uniqueitemidentifier and
OverDueLoan.loanstartdate = eachrow.loanstartdate and OverDueLoan.FineResolvedDate is null;
      returnLoan(eachrow.UniqueItemIdentifier);
      end loop;
      Select members.AccountStateId into memberstatus from members where members.MemberId = membernum;
      if memberstatus = 2 then
      update members set members.AccountStateId = 1 where members.MemberId = membernum;
      end if;

   end;
/
```

Test statement:

```
execute payFines(120381301);
select * from "Loans History";
```



```
execute payFines(112318907);
select * from "Loans History";
```

```
Statement processed.
Item 11 returned successfully!
Item 8 returned successfully!
```

User number	User	Item	Start Date	Return Date	Was Overdue	Fine amount paid
112318907	Ralph Finnes	UII:11 -- "Extreme Programming in Practice" by James W. Newkirk, Robert C. Martin	11 Nov 2021 08:34	17 Dec 2021 02:19	True	21
112318907	Ralph Finnes	UII:8 -- "Clean Code: A Handbook of Agile Software Craftsmanship" by Robert C. Martin	11 Nov 2021 06:14	17 Dec 2021 02:19	True	33
160591543	David Nitu	UII:1 -- "Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev	17 Dec 2021 01:10	17 Dec 2021 02:02	False	-

6.4 Loan Items

The loanItem procedures takes in the parameters UII and memberId. The procedures checks the availability of the item to be loaned, and populates the loan table with appropriate member to loan associations. This will be displayed in the functionality section.

```
Create or replace procedure loanItem
 (UII number, membernum number)
 is
 lastestReturnDate timestamp;
 memberstatus varchar2(20);
 resourcesLoaned number;
 maxResources number;
 libraryloc number;
 libraryAccess varchar2(10);
 libName varchar2(50);
  begin
    lastestReturnDate := getLatestLoanStartDate(UII);
    select StateValue into memberstatus from members
    left join AccountState on members.AccountStateId = AccountState.AccountStateId where members.memberid = membernum;
    select count(*) into resourcesLoaned from loan where loan.memberid = membernum and loan.loanreturndate is null;
    select maxNoOfResources into maxResources from members
    left join AccountType on members.AccountTypeId = AccountType.AccountTypeId where members.memberid = membernum;
         Select libraryid into libraryloc from uniqueitem left join shelf_location on uniqueitem.locationid = shelf_location.shelfid where
         uniqueitem.uniqueitemidentifier = UII;
    Select libraryName into libName from lbry where lbry.libraryid = libraryloc;
    Select accessid into libraryAccess from Lbry_Access where Lbry_Access.memberId = membernum and Lbry_Access.libraryid = libraryloc
    If (memberstatus like 'Active') then
      If (libraryAccess = 'True') Then
        If (getLoanDuration(UII) > interval '0' day) Then
          If (lastestReturnDate is null) Then
            If (resourcesLoaned < maxResources) then
              Insert into Loan (LoanStartDate, MemberId, UniqueItemIdentifier) values(current_timestamp, membernum, UII);
            Else dbms_output.put_line('This member has already booked the maximum number of resources allowed!');
            End if;
          Else dbms_output.put_line('This resource is currently booked by another member!');
          End if;
        Else dbms_output.put_line('This resource can only be used within the library!');
        End if;
      Else dbms_output.put_line('This member cannot book items from the ' || libName || '!');
      End if;
    Else dbms_output.put_line('This member has a ' || lower(memberstatus) || ' account!');
    End if;
  end;
/
```

6.5 Update Member Status

The updateMemberStatus procedure checks if a member meets the criteria to be suspended and suspends them if appropriate.

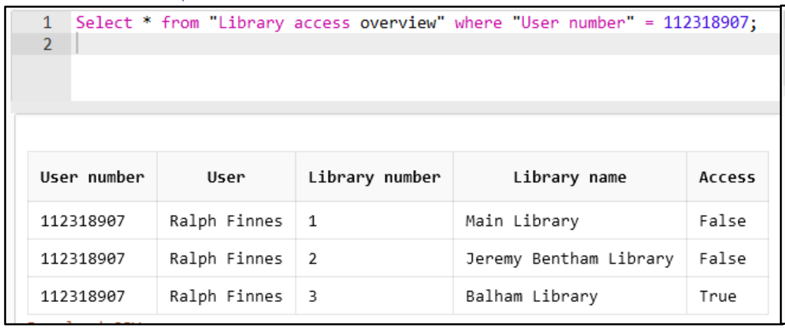
This procedure is run within the dailyUpdate procedure.

```
Create or replace procedure updateMemberStatus
  is
  fine number;
  memberstatus number;
  cursor overdueloans
  is
    select MemberId, Sum(calculateFine(MemberId, uniqueitemidentifier, loanstartdate)) totalFines from overdueloan where fineresolveddate is null
group by memberid;
    begin
        for eachrow in overdueloans
        loop
            Select members.AccountStateId into memberstatus from members where members.MemberId = eachrow.MemberId;
            if (eachrow.totalFines > 10) and (memberstatus = 1) then
            update members set members.AccountStateId = 2 where members.MemberId = eachrow.MemberId;
            end if;
        end loop;
    end;
/
```

6.6 Grant library access

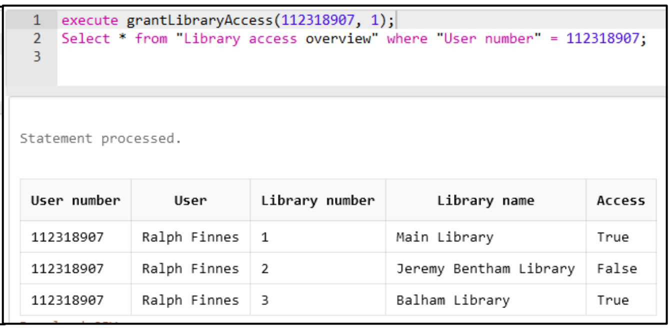
The grantLibraryAccess procedure grants access of a specific library to a specific member.

```
Create or replace procedure grantLibraryAccess
                  (membernum IN number, librarynum IN number)
        IS
    accesslvl varchar2(10);
    begin
        Select accessid into accesslvl from lbry_access where memberid = membernum and  libraryid = librarynum;
        if accesslvl = 'False' then
            Update lbry_access set accessid = 'True' where memberid = membernum and  libraryid = librarynum;
        else
            dbms_output.put_line('This member already has access to this library.');
            end if;
    end;
/
```



```
Select * from "Library access overview" where "User number" = 112318907;
```

User number	User	Library number	Library name	Access
112318907	Ralph Finnes	1	Main Library	False
112318907	Ralph Finnes	2	Jeremy Bentham Library	False
112318907	Ralph Finnes	3	Balham Library	True



```
execute grantLibraryAccess(112318907, 1);
Select * from "Library access overview" where "User number" = 112318907;
```

```
Statement processed.
```

User number	User	Library number	Library name	Access
112318907	Ralph Finnes	1	Main Library	True
112318907	Ralph Finnes	2	Jeremy Bentham Library	False
112318907	Ralph Finnes	3	Balham Library	True

BEFORE **AFTER**

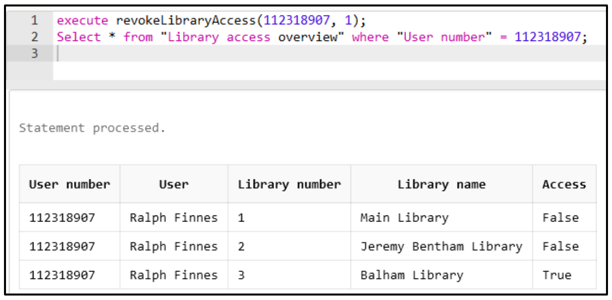
6.6 Revoke library access

The revokeLibraryAccess procedure revokes access of a specific library to a specific member.

```
Create or replace procedure revokeLibraryAccess
            (membernum IN number, librarynum IN number)
    IS
  accesslvl varchar2(10);
  BEGIN
    Select accessid into accesslvl from lbry_access where memberid = membernum and  libraryid = librarynum;
    if accesslvl = 'True' then
      Update lbry_access set accessid = 'False' where memberid = membernum and  libraryid = librarynum;
    else
      dbms_output.put_line('This member already has access to this library.');
      end if;
  END;
/
```

Test statement:

```
execute revokeLibraryAccess(112318907, 1);
Select * from "Library access overview" where "User number" = 112318907;
```



```
execute revokeLibraryAccess(112318907, 1);
Select * from "Library access overview" where "User number" = 112318907;

Statement processed.
```

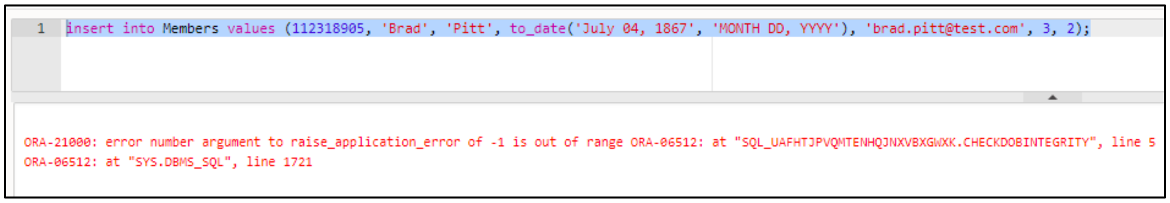
User number	User	Library number	Library name	Access
112318907	Ralph Finnes	1	Main Library	False
112318907	Ralph Finnes	2	Jeremy Bentham Library	False
112318907	Ralph Finnes	3	Balham Library	True

7. TRIGGERS

7.1 Check date of birth

The checkDobIntegrity trigger checks the integrity of the date of birth (DoB) attribute value in the Members table. The member's date of birth must be after Jan 2nd, 1903 (currently oldest living person) and over 15 years old.

```
Create or replace trigger checkDobIntegrity
  before insert or update on Members
    for each row
  begin
  if( :new.Dob < date '1903-01-02' or
    :new.Dob > sysdate - 365*15 )
  then
    raise_application_error(-00001, 'The member date of birth must be after Jan 2nd, 1903 and over 15 years old');
  end if;
end;
/
```



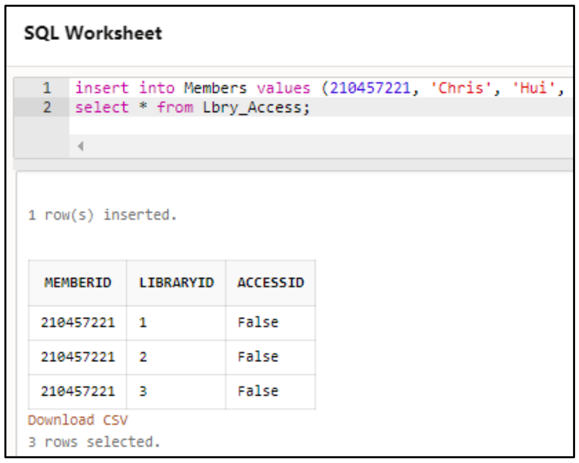
```
insert into Members values (112318905, 'Brad', 'Pitt', to_date('July 04, 1867', 'MONTH DD, YYYY'), 'brad.pitt@test.com', 3, 2);

ORA-21000: error number argument to raise_application_error of -1 is out of range ORA-06512: at "SQL_UAFHTJPVQMTENHQJNXVBXGWXK.CHECKDOBINTEGRITY", line 5
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

7.2 Populate library access

The populateLibraryAccess trigger functions such that when a new member is inserted into the Members table, their access to all libraries is initially set to False.

```
Create or replace trigger populateLibraryAccess
    after insert
    on Members
    for each row
declare
    cursor libraries
        IS
    select libraryid from lbry;
begin
    for lib in libraries
    loop
    Insert into lbry_access values(:new.memberid, lib.libraryid, 'False');
    end loop;
end;
/
```


SQL Worksheet

```
insert into Members values (210457221, 'Chris', 'Hui',
select * from Lbry_Access;
```

```
1 row(s) inserted.
```

MEMBERID	LIBRARYID	ACCESSID
210457221	1	False
210457221	2	False
210457221	3	False

Download CSV
3 rows selected.

7.3 Max loan notification

The maxLoanNotification trigger sends a notification to a user when they have loaned the maximum allowed amount of resources associated with their member type. Additionally, it displays a notification when a member loans an item, showing the number of resources they have currently loaned. This will be displayed in the functionality section.

```
create or replace trigger maxLoanNotification
    before insert
    on Loan
    for each row
declare
    maxResources number(2);
    resourcesLoaned number;
begin
    -- get user's max possible number of resources
    select maxNoOfResources into maxResources from Members
    left join AccountType on Members.AccountTypeId = AccountType.AccountTypeId where Members.Memberid = :new.MemberId;
    -- get user's current number of resources loaned
    select count(*) into resourcesLoaned from Loan where Loan.MemberId = :new.MemberId and Loan.LoanReturnDate is null;
    if (resourcesLoaned + 1 = maxResources) then
        dbms_output.put_line('Item succesfully loaned! Member has now loaned the maximum amount of resources!');
    else
        dbms_output.put_line('Item succesfully loaned! Member has now loaned ' || (resourcesLoaned + 1) || ' resources!');
    end if;
end;
/
```

7.4 Loan expiry notification

The loan expiry notification trigger informs the user when a loan has expired and needs to be returned. This will be displayed in the functionality section. This will also be displayed in the functionality section.

```
Create or replace trigger sendLoanExpiryNotification
    after insert
    on overdueloan
    for each row
declare
username varchar2(100);
res varchar2(500);
begin
    Select (firstname || ' ' || lastname) into username from members where members.memberid = :new.memberid;
    Select "Resource" into res from "Items overview" where "Item identifier" = :new.uniqueitemidentifier;
    dbms_output.put_line('Hi ' || username || ' (member id: ' || :new.memberid || '). Your loan for ' || res || ' has expired. If it is not returned within 24 hours, a charge will be incurred');
end;
/
```

8. VIEWS

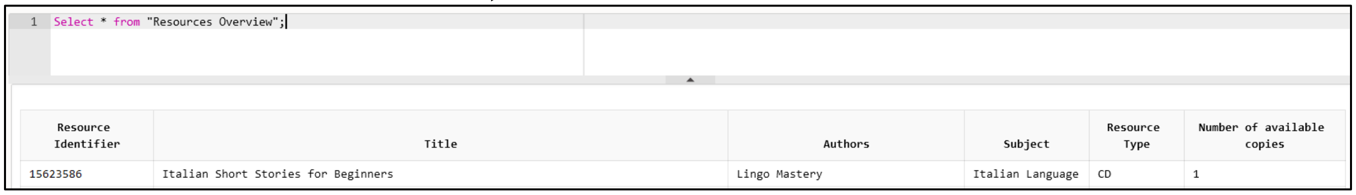
8.1 Resources Overview

Create Resources Overview view that displays the resources available within the library and some other helpful attributes.

```
Create or replace view "Resources Overview" as
    Select * from
        (Select ISBN "Resource Identifier", Btitle "Title", getBookAuthor(ISBN) "Authors", SubjectName "Subject", getResType(ISBN) "Resource Type", numberOfCopies(ISBN) "Number of available copies" from Book
            left join Subject on Subject.subjectnumber = Book.subjectnumber)
    union
    Select * from
        (select ISSN, Mtitle, getMediaAuthor(ISSN), SubjectName "Subject", getResType(ISSN), numberOfCopies(ISSN) FROM Media
            left join Subject on Subject.subjectnumber = Media.subjectnumber);
```

Test statement:

```
Select * from "Resources Overview";
```



```
Select * from "Resources Overview";
```

Resource Identifier	Title	Authors	Subject	Resource Type	Number of available copies
15623586	Italian Short Stories for Beginners	Lingo Mastery	Italian Language	CD	1

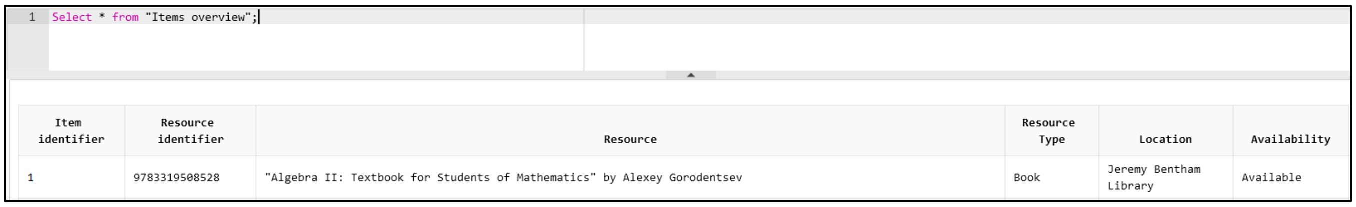
8.2 Items Overview

Create Items Overview view that displays resources found in the library and their availability.

```
Create or replace view "Items overview" as
  Select uniqueitem.uniqueitemidentifier "Item identifier", nvl(uniqueitem.isbn, uniqueitem.issn) "Resource identifier",
    ('"' || "Title" || '"' || ' by ' || "Authors") "Resource", "Resource Type", locationDetails.libraryName "Location",
    checkAvailability(uniqueitem.uniqueitemidentifier) "Availability" from uniqueitem
    left join "Resources Overview" on nvl(uniqueitem.isbn, uniqueitem.issn) = "Resources Overview"."Resource Identifier"
    left join (Select libraryName, shelfid from shelf_location
      left join lbry on shelf_location.libraryid = lbry.libraryid) locationDetails on locationId = locationDetails.shelfid order by "Item identifier";
```

Test statement:

Select * from "Items overview";



```
Select * from "Items overview";
```

Item identifier	Resource identifier	Resource	Resource Type	Location	Availability
1	9783319508528	"Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev	Book	Jeremy Bentham Library	Available

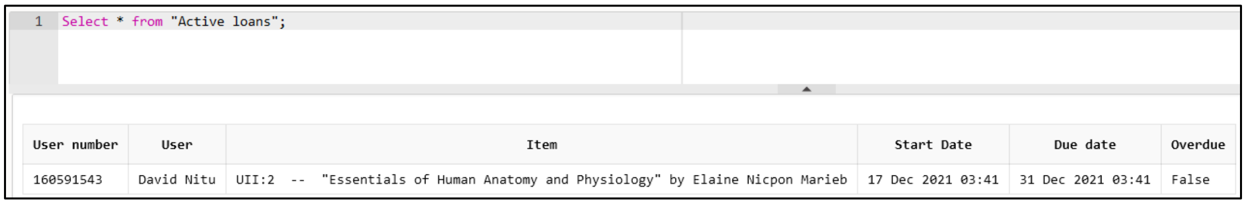
8.3 Active loans

Create Active Loans view that displays active loans.

```
Create or replace view "Active loans" as
select loan.memberid "User number", (firstname || ' ' || lastname) "User", ('UII:' || loan.uniqueItemIdentifier || '  --  ' ||"Resource") "Item",
  to_char(loanStartDate,'DD Mon YYYY HH24:MI') "Start Date", to_char(loan.loanstartDate + getLoanDuration(loan.uniqueItemIdentifier), 'DD Mon YYYY HH24:MI') "Due date",
  (Case when ((current_timestamp - LoanStartDate) > getLoanDuration(UniqueItemIdentifier)) then 'True' else 'False' end) "Overdue" from loan
  left join members on loan.memberid = members.memberid
  left join "Items overview" on loan.uniqueItemIdentifier = "Items overview"."Item identifier"
  where loanReturnDate is null;
```

Test statement:

Select * from "Active loans";



```
Select * from "Active loans";
```

User number	User	Item	Start Date	Due date	Overdue
160591543	David Nitu	UII:2 -- "Essentials of Human Anatomy and Physiology" by Elaine Nicpon Marieb	17 Dec 2021 03:41	31 Dec 2021 03:41	False

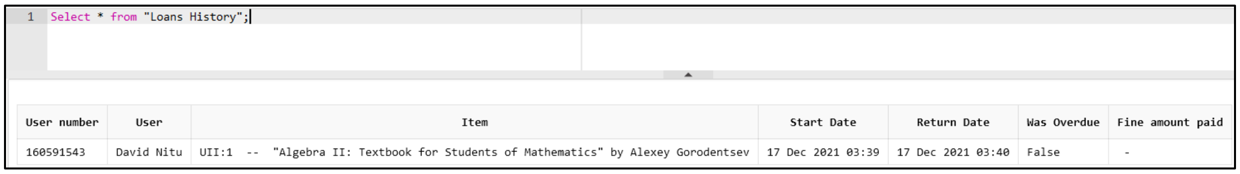
8.4 Loans History

Create Loans History view that displays the loan history.

```
Create or replace view "Loans History" AS
select loan.memberid "User number", (firstname || ' ' || lastname) "User", ('UII:' || loan.uniqueItemIdentifier || '  --  ' ||"Resource") "Item",
   to_char(loan.loanStartDate,'DD Mon YYYY HH24:MI') "Start Date", to_char(loan.loanreturnDate,'DD Mon YYYY HH24:MI') "Return Date",
   (Case when ((loan.loanReturnDate - loan.LoanStartDate) > getLoanDuration(loan.UniqueItemIdentifier)) then 'True' else 'False' end) "Was Overdue"
   ,overdueloan.loantotalFine "Fine amount paid"
   from loan
   left join members on loan.memberid = members.memberid
   left join "Items overview" on loan.uniqueItemIdentifier = "Items overview"."Item identifier"
   left join overdueloan on loan.uniqueItemIdentifier = overdueloan.uniqueItemIdentifier
   and (loan.memberid = overdueloan.memberid) and (loan.loanstartdate = overdueloan.loanstartdate)
   where loanReturnDate is not null;
```

Test statement:

Select * from "Loans History";



```
Select * from "Loans History";
```

User number	User	Item	Start Date	Return Date	Was Overdue	Fine amount paid
160591543	David Nitu	UII:1 -- "Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev	17 Dec 2021 03:39	17 Dec 2021 03:40	False	-

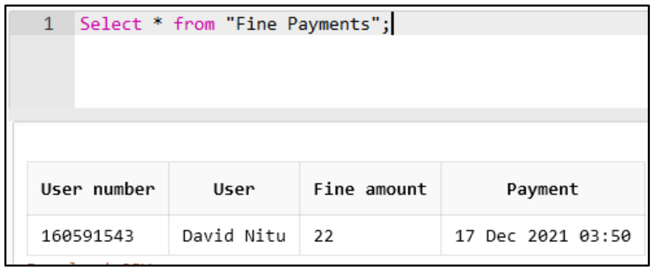
8.5 Fine Payments

Create Fine Payments view that displays payments by user.

```
Create or replace view "Fine Payments" as
   select overdueloan.memberid "User number", (firstname || ' ' || lastname) "User", nvl(sum(loantotalfine),
      calculateTotalFine(overdueloan.memberid)) "Fine amount", nvl(to_char(fineresolveddate, 'DD Mon YYYY HH24:MI'), 'Outstanding') "Payment"
   from overdueloan left join members on overdueloan.memberid = members.memberid
      group by to_char(fineresolveddate, 'DD Mon YYYY HH24:MI'), overdueloan.memberid, (firstname || ' ' || lastname)
      order by "Payment" DESC;
```

Test statement:

Select * from "Fine Payments";



```
Select * from "Fine Payments";
```

User number	User	Fine amount	Payment
160591543	David Nitu	22	17 Dec 2021 03:50

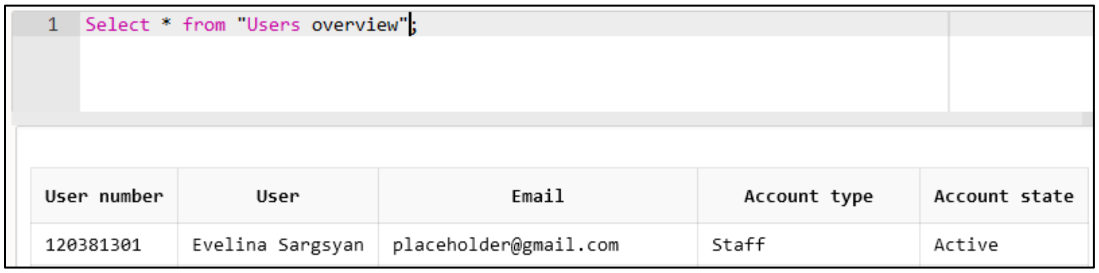
8.6 Users overview

Create Users overview view that displays user details

```
Create or replace view "Users overview" AS
   select memberid "User number", (firstname || ' ' || lastname) "User", email "Email", typevalue "Account type", statevalue "Account state" from members
   left join accountstate on members.accountstateid = accountstate.accountstateid
   left join accounttype on members.accounttypeid = accounttype.accounttypeid;
```

Test statement:

Select * from "Users overview";



```
Select * from "Users overview";
```

User number	User	Email	Account type	Account state
120381301	Evelina Sargsyan	placeholder@gmail.com	Staff	Active

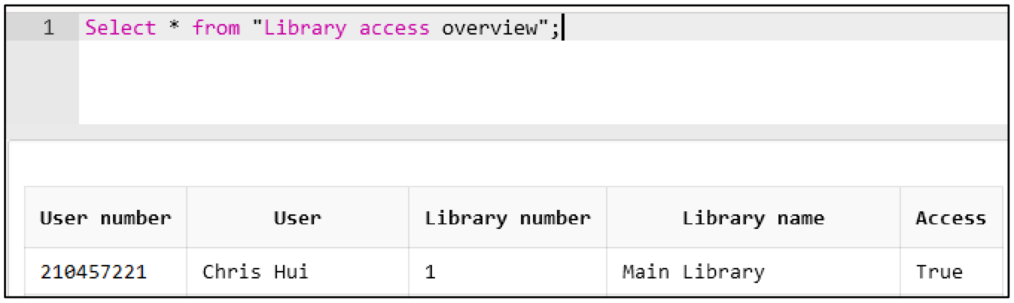
8.7 Library access overview

Create Library access overview that displays users and their access level to different libraries

```
Create or replace view "Library access overview" AS
   select lbry_access.memberid "User number", (firstname || ' ' || lastname) "User", lbry_access.libraryid "Library number", libraryname "Library name", accessid "Access" from lbry_access
   left join members on members.memberid = lbry_access.memberid
   left join lbry on lbry_access.libraryid = lbry.libraryid;
```

Test statement:

Select * from "Library access overview";



```
Select * from "Library access overview";
```

User number	User	Library number	Library name	Access
210457221	Chris Hui	1	Main Library	True

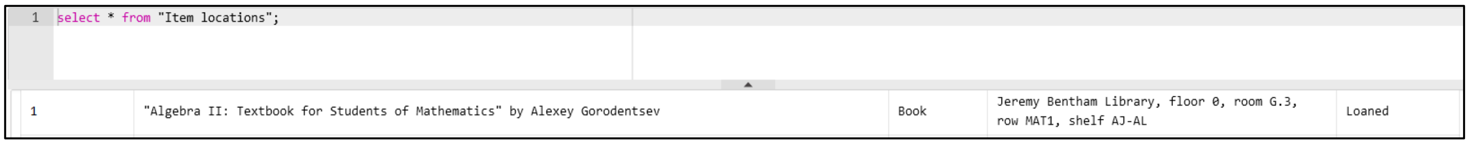
8.8 Item locations

Create Item locations view that displays the locations of specific items within the library

```
Create or replace view "Item locations" AS
   Select uniqueitem.uniqueitemidentifier "Item identifier", ('"' || "Title" || '"' || ' by ' || "Authors") "Resource", "Resource Type",
   (locationDetails.libraryName || ', floor ' || locationDetails.floor || ', room ' || locationDetails.room || ', row ' || locationDetails.rownumber || ', shelf ' || locationDetails.shelf) "Location",
      checkAvailability(uniqueitem.uniqueitemidentifier) "Availability" from uniqueitem
      left join "Resources Overview" on nvl(uniqueitem.isbn, uniqueitem.issn) = "Resources Overview"."Resource Identifier"
      left join (Select libraryName, floor, room, rownumber, shelf,  shelfid from shelf_location
         left join lbry on shelf_location.libraryid = lbry.libraryid) locationDetails on locationId = locationDetails.shelfid order by "Item identifier";
```

Test statement:

```
Select * from "Item locations";
```



```
select * from "Item locations";
```

```
1    "Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev    Book    Jeremy Bentham Library, floor 0, room G.3,    Loaned
                                                                                         row MAT1, shelf AJ-AL
```

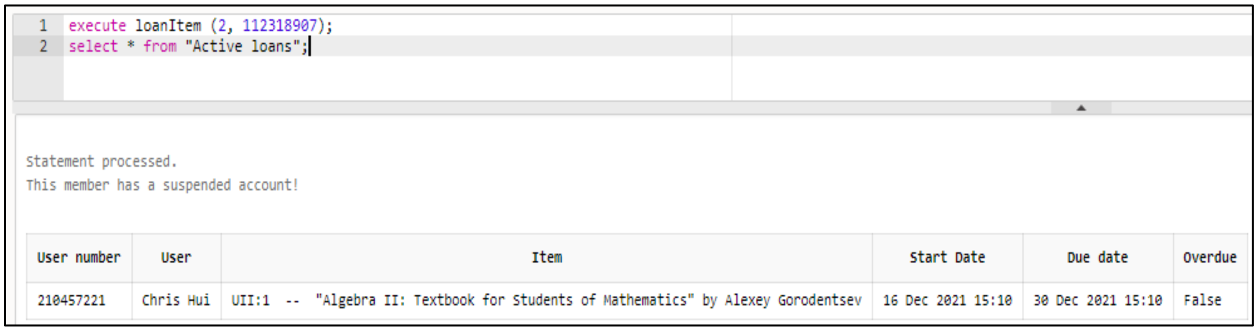
9. FUNCTIONALITY

For the purposes of this library system the users (library members) and library staff will only be able to interact with the database using procedures in order to minimize the insert errors and avoid changing or deleting sensible data from the tables. So far, we have implemented functionality for loaning an item, returning a loan, pay user fines and grant access to library. The loanItem procedure can be executed in order to record a new loan in the loan table. The procedure takes in an unique item identifier followed by a member id and if it passes the checks, it adds the new record with a start datetime value of the systems current date and time. Because Oracle Live SQL did not allow us to change the time zone, the dates will be displayed using US/Pacific time zone. If we would have privileges to change the database settings the following statement can be run to change the time zone to Europe/London: ALTER DATABASE SET TIME_ZONE='Europe/London';

To execute the loanItem procedure the following statement can be run:
EXECUTE loanItem(uniqueitemidentifier, memberid);

Where uniqueitemidentifier must be a valid item within the unique item table and memberid must refer to a valid member within the member's table. This procedure will return the loaned item, logging the server's current timestamp as the loan's return date.The procedure will process a series of *if* statements as below:

- The library member must be active, and not suspended or a closed account.

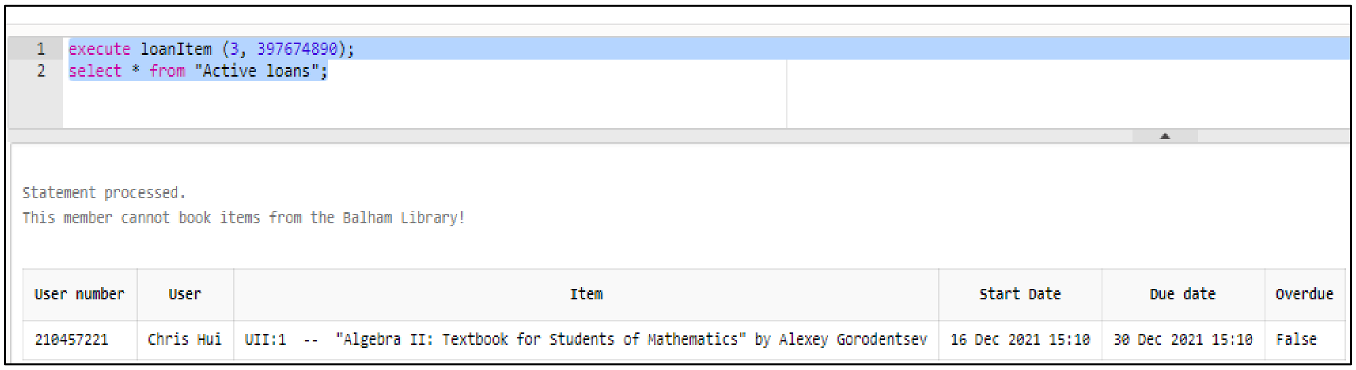


```
execute loanItem (2, 112318907);
select * from "Active loans";
```

```
Statement processed.
This member has a suspended account!
```

User number	User	Item	Start Date	Due date	Overdue
210457221	Chris Hui	UII:1 -- "Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev	16 Dec 2021 15:10	30 Dec 2021 15:10	False

- The library member must have library access privilege to the library where the resource is located in.

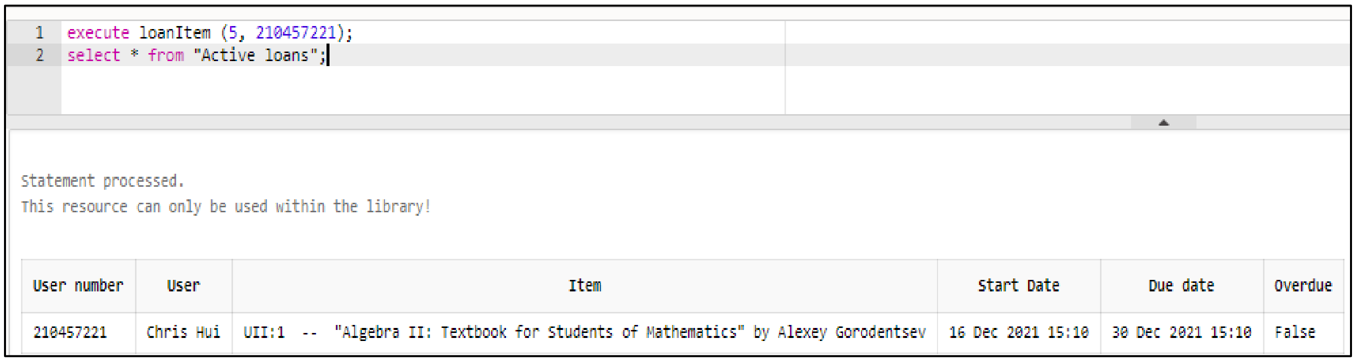


```
execute loanItem (3, 397674890);
select * from "Active loans";
```

```
Statement processed.
This member cannot book items from the Balham Library!
```

User number	User	Item	Start Date	Due date	Overdue
210457221	Chris Hui	UII:1 -- "Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev	16 Dec 2021 15:10	30 Dec 2021 15:10	False

- The specified resource item must have a loan duration of more than 0 day. As a 0 day loan duration indicates that the resource must be used within the library only.

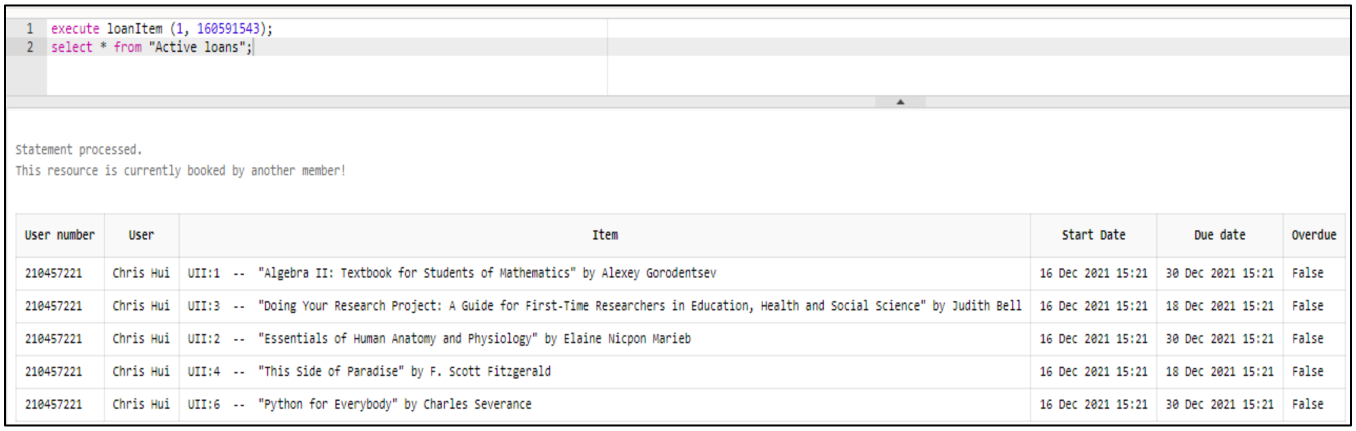


```
execute loanItem (5, 210457221);
select * from "Active loans";
```

```
Statement processed.
This resource can only be used within the library!
```

User number	User	Item	Start Date	Due date	Overdue
210457221	Chris Hui	UII:1 -- "Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev	16 Dec 2021 15:10	30 Dec 2021 15:10	False

- The specific resource must not already be loaned.

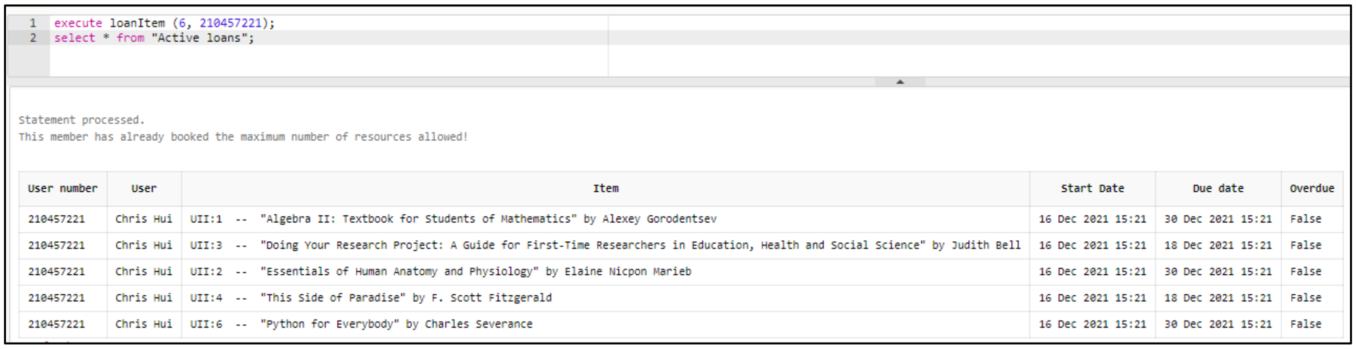


```
execute loanItem (1, 160591543);
select * from "Active loans";
```

```
Statement processed.
This resource is currently booked by another member!
```

User number	User	Item	Start Date	Due date	Overdue
210457221	Chris Hui	UII:1 -- "Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev	16 Dec 2021 15:21	30 Dec 2021 15:21	False
210457221	Chris Hui	UII:3 -- "Doing Your Research Project: A Guide for First-Time Researchers in Education, Health and Social Science" by Judith Bell	16 Dec 2021 15:21	18 Dec 2021 15:21	False
210457221	Chris Hui	UII:2 -- "Essentials of Human Anatomy and Physiology" by Elaine Nicpon Marieb	16 Dec 2021 15:21	30 Dec 2021 15:21	False
210457221	Chris Hui	UII:4 -- "This Side of Paradise" by F. Scott Fitzgerald	16 Dec 2021 15:21	18 Dec 2021 15:21	False
210457221	Chris Hui	UII:6 -- "Python for Everybody" by Charles Severance	16 Dec 2021 15:21	30 Dec 2021 15:21	False

- The number of resources already loaned by the library member must be less than the maximum of resources that can be loaned by that member. This displays the maxLoanNotification trigger.

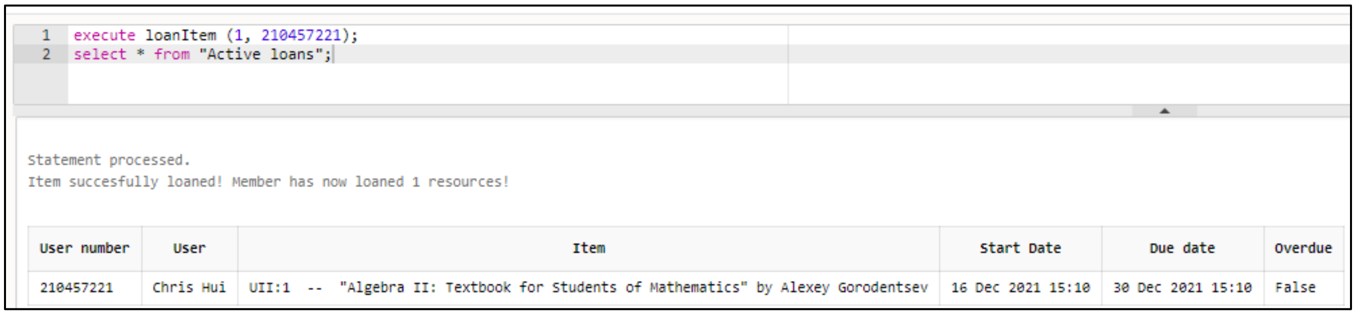


```
execute loanItem (6, 210457221);
select * from "Active loans";
```

```
Statement processed.
This member has already booked the maximum number of resources allowed!
```

User number	User	Item	Start Date	Due date	Overdue
210457221	Chris Hui	UII:1 -- "Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev	16 Dec 2021 15:21	30 Dec 2021 15:21	False
210457221	Chris Hui	UII:3 -- "Doing Your Research Project: A Guide for First-Time Researchers in Education, Health and Social Science" by Judith Bell	16 Dec 2021 15:21	18 Dec 2021 15:21	False
210457221	Chris Hui	UII:2 -- "Essentials of Human Anatomy and Physiology" by Elaine Nicpon Marieb	16 Dec 2021 15:21	30 Dec 2021 15:21	False
210457221	Chris Hui	UII:4 -- "This Side of Paradise" by F. Scott Fitzgerald	16 Dec 2021 15:21	18 Dec 2021 15:21	False
210457221	Chris Hui	UII:6 -- "Python for Everybody" by Charles Severance	16 Dec 2021 15:21	30 Dec 2021 15:21	False

- If the member and resource item passes above *if* conditions, a loan for that resource will be made.

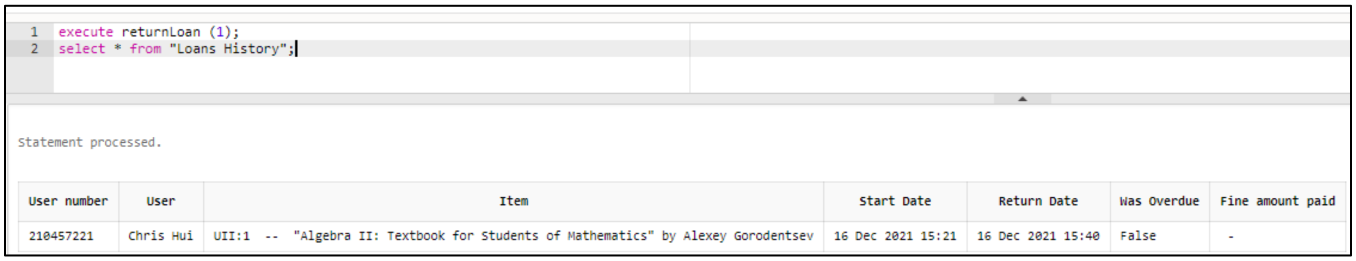


```
execute loanItem (1, 210457221);
select * from "Active loans";
```

```
Statement processed.
Item succesfully loaned! Member has now loaned 1 resources!
```

User number	User	Item	Start Date	Due date	Overdue
210457221	Chris Hui	UII:1 -- "Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev	16 Dec 2021 15:10	30 Dec 2021 15:10	False

- To return a loaned item, the following returnLoan procedure can be run:
  EXECUTE returnLoan(uniqueitemidentifier);



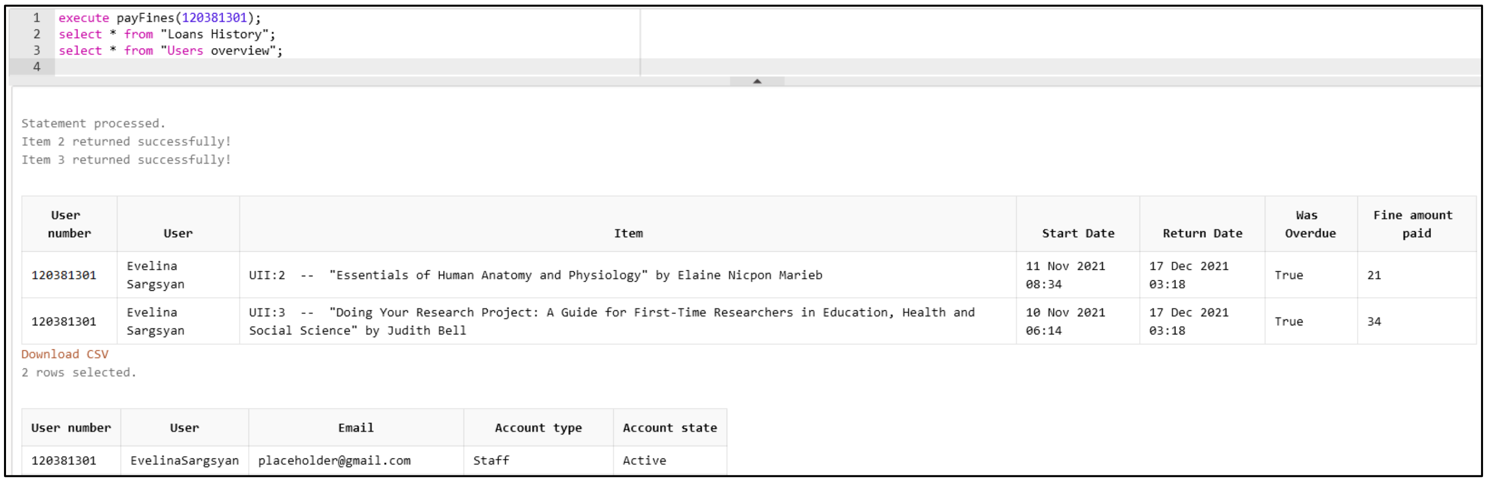
```
execute returnLoan (1);
select * from "Loans History";
```

```
Statement processed.
```

User number	User	Item	Start Date	Return Date	Was Overdue	Fine amount paid
210457221	Chris Hui	UII:1 -- "Algebra II: Textbook for Students of Mathematics" by Alexey Gorodentsev	16 Dec 2021 15:21	16 Dec 2021 15:40	False	-

- To pay overdue loan fines for a member, the following procedure can be run:
  EXECUTE payFines(memberid);

This procedure will resolve all overdue loans associated with overdue loan items for the member. A member will pay all of their outstanding fines at once.
Because a member must first return all overdue loans in order to pay their fines, the payFines procedure will also call the returnLoan procedure.



```
execute payFines(120381301);
select * from "Loans History";
select * from "Users overview";
```

```
Statement processed.
Item 2 returned successfully!
Item 3 returned successfully!
```

User number	User	Item	Start Date	Return Date	Was Overdue	Fine amount paid
120381301	Evelina Sargsyan	UII:2 -- "Essentials of Human Anatomy and Physiology" by Elaine Nicpon Marieb	11 Nov 2021 08:34	17 Dec 2021 03:18	True	21
120381301	Evelina Sargsyan	UII:3 -- "Doing Your Research Project: A Guide for First-Time Researchers in Education, Health and Social Science" by Judith Bell	10 Nov 2021 06:14	17 Dec 2021 03:18	True	34

```
Download CSV
2 rows selected.
```

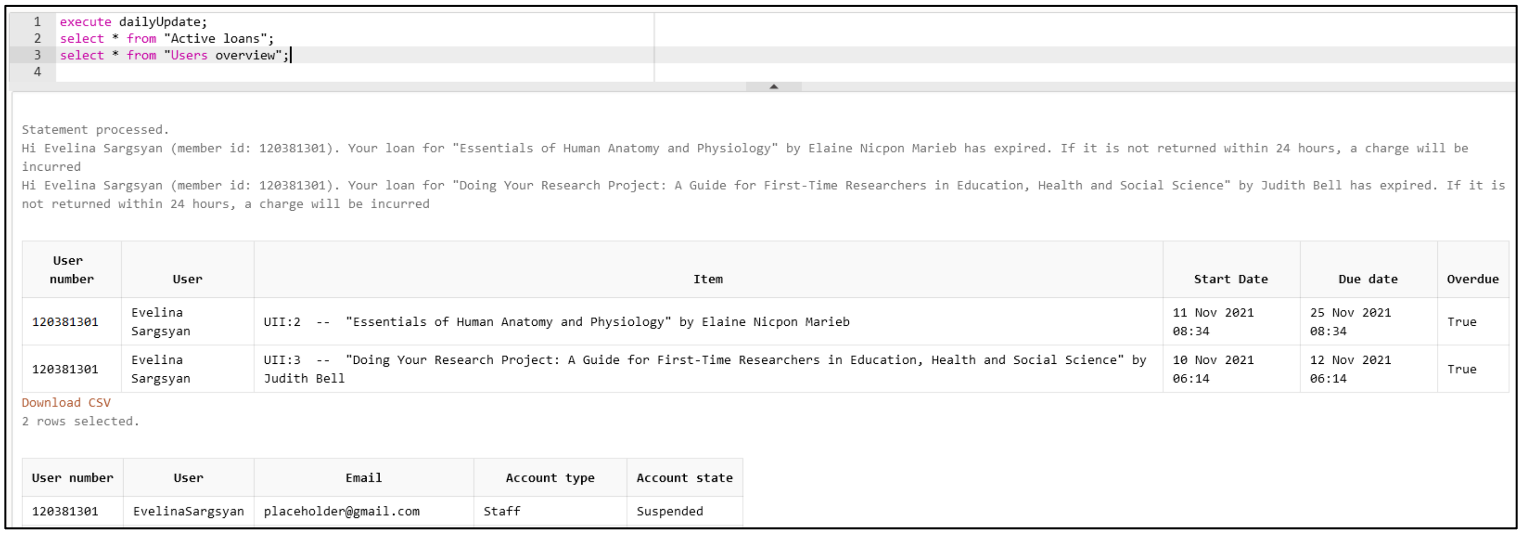
User number	User	Email	Account type	Account state
120381301	EvelinaSargsyan	placeholder@gmail.com	Staff	Active

The Create dailyUpdate procedure. This simulates a daily update and replaces the functionality that dbms_scheduler would provide, as it cannot be run on Oracle SQL Live. The user must therefore execute this procedure manually. The procedure uses some of the functionality that we have implemented in previous sections and performs daily updates based on the following parameters:

- New loans that are overdue will be added to the OverdueLoan table
- Members with fines exceeding the maximum allowed amount will be suspended
- Members with loans due within 1 day will be notified

```
Create or replace procedure dailyUpdate
 is
 cursor memberfromloans
 is
   select distinct MemberId  from loan where loanreturndate is null;
 cursor distinctloans
 is
   select "User number", "User", "Item", "Due date" from "Active loans" where "Overdue" = 'False';
   begin
       for eachrow in memberfromloans
       loop
           updateloans(eachrow.memberid);
       end loop;
       updatememberstatus;
       for eachrow in distinctloans
       loop
       if (((sysdate - to_date(eachrow."Due date", 'dd:mm:yyyy hh24:mi')) >= -1)) then
       dbms_output.put_line('Hi ' || eachrow."User" || ' (member id: ' || eachrow."User number" || '). Your loan for ' || regexp_substr(eachrow."Item",
' "[^,]+', 1, 1) || ' will expire in less than a day!');
       end if;
       end loop;

   end;
/
```



```
execute dailyUpdate;
select * from "Active loans";
select * from "Users overview";
```

```
Statement processed.
Hi Evelina Sargsyan (member id: 120381301). Your loan for "Essentials of Human Anatomy and Physiology" by Elaine Nicpon Marieb has expired. If it is not returned within 24 hours, a charge will be incurred
Hi Evelina Sargsyan (member id: 120381301). Your loan for "Doing Your Research Project: A Guide for First-Time Researchers in Education, Health and Social Science" by Judith Bell has expired. If it is not returned within 24 hours, a charge will be incurred
```

User number	User	Item	Start Date	Due date	Overdue
120381301	Evelina Sargsyan	UII:2 -- "Essentials of Human Anatomy and Physiology" by Elaine Nicpon Marieb	11 Nov 2021 08:34	25 Nov 2021 08:34	True
120381301	Evelina Sargsyan	UII:3 -- "Doing Your Research Project: A Guide for First-Time Researchers in Education, Health and Social Science" by Judith Bell	10 Nov 2021 06:14	12 Nov 2021 06:14	True

Download CSV
2 rows selected.

User number	User	Email	Account type	Account state
120381301	EvelinaSargsyan	placeholder@gmail.com	Staff	Suspended

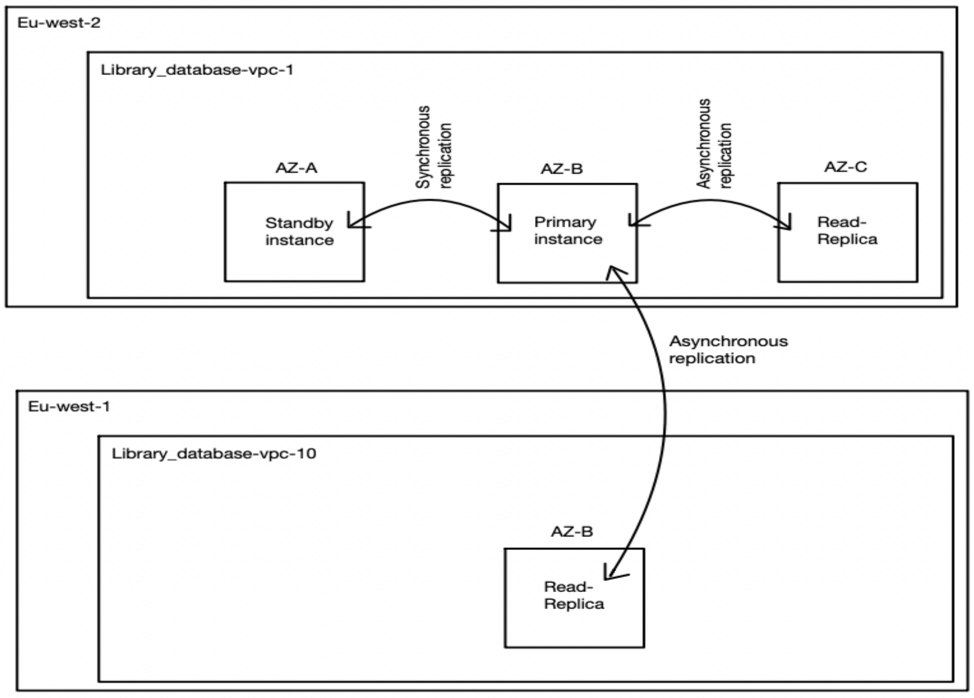
10. IMPLEMENTATION, SECURITY AND CONCLUSIONS

The Library Database will utilise the Amazon Web Services (AWS) Relational Database Service (RDS) for Oracle in production. The AWS RDS is a database-as-a-service (DBaas) product (or more accurately Database*Server*-as-a-service), that utilises cloud computing to provide the infrastructure for the database, to be used in conjunction with other AWS products including networking (Virtual Private Cloud), Storage (Elastic Block Storage, S3), and security (Key Management Service, Identity and Access Management).

At the outset, a primary database instance will be set up in a Virtual Private Cloud (VPC), 'Library_database-vpc-1', within the eu-west-2 (London) region of AWS, inside Availability Zone B (AZ-B). The primary database instance will support read/write operations and will be primary connection endpoint during normal operation.

In an effort to ensure high availability, durability, and protection, a Multi-Availability Zone (Multi-AZ) deployment will be utilised. MultiAZ is the provision of a standby replica, where the data on the standby instance is kept in sync with the primary instance. The standby instance will be deployed in a different availability zone, which enhances the resilience of the database. In the event of a database instance failure or Availability Zone failure, a failover automatically occurs which takes approximately 60-120s.

To further improve performance, RDS Read-Replicas will also be utilised. Read-replicas provide asynchronous replication with the primary instance. 2x further read-replica instances will be provisioned, one in AZ-C within the same region (eu-west-2) and vpc (Library_database-vpc-1) as the primary instance, and the second in a VPC at a separate region eu-west-1 (Ireland), 'Library_database-vpc-10'. Read queries (e.g. query available copies of a book) will be routed to the two additional provisioned read-replica instances, thus reducing the load on the primary database instance. The architecture of the RDS system is shown below.



Recovery Point Objective (RPO) is defined as the time between last backup and a data-loss incident, which indicates the maximum amount of data that can be loss. By implementing a Multi-AZ and Read-Replica deployment in parallel, the RPO value is lowered. In addition, automatic RDS backups will be initialised. AWS S3 object storage will be used to store the transaction logs every 5 minutes, allowing for point in time recovery.

The Library Database's data security concerns surrounding authentication and authorisation (i.e. how users can log into the database, and how access is controlled) will be addressed by utilising the Identity and Access Management (IAM) service in AWS. Through IAM, users accessing the database will have different access privileges, categorised under IAM roles. Appropriate IAM identity policies, a set of security statement that grants/deny access to specific features, will be attached to these IAM roles. A brief overview of IAM roles are as follows:

- Administrator role - Full access privileges.
- Library Staff role - Read and write privileges where appropriate, e.g. ability to create new data entry for new resources.
- Academic staff and student role - Read privileges.
- Temporary/external role - Temporary/external users, for example visiting students, can assume this role to temporarily gain read privileges.

The following IAM identity policies will be attached to the relevant IAM roles:

- AmazonRDSReadOnlyAccess - Grants read-only access to the RDS instance resources.
- AmazonRDSFullAccess - Grants full access to the RDS instance resources.

Encryption in transit between clients and the RDS instance is supported using Secure Socket Layer (SSL) and Transport Layer Security (TLS). SSL/TLS connections encrypts the data that moves between clients and the database instance.

In regards to encryption at rest, RDS data of the primary instance, standby instance, read-replicas, and backup snapshots stored in the Elastic Block Storage (EBS) volume is encrypted, using Data Encryption Keys (DEK) managed by the AWS Key Management Service (KMS). AWS KMS uses hardware security modules that have been validated under FIPS 140-2 level 2 security standard.

In addition, RDS Oracle also support Transparent Data Encryption (TDE), so that RDS data is encrypted using AES and 3DES encryption algorithm. This is integrated with the AWS CloudHSM, a "single tenant" Hardware Security Module (HSM). Through the CloudHSM, the administrator will be able to securely generate, store, and manage the cryptographic keys. CloudHSM is fully FIPS 140-2 level 3 overall security standard compliant. CloudHSM will be accessed with industry standard APIs e.g. PKCS#11, Java Cryptography Extensions (JCE), and Microsoft CryptoNG (CNG).

To ensure compliance with data processing elements in the UK Data Protection Act 2018, personal data related to the users of the Library Database (e.g. personal addresses, contact information) will be used fairly, lawfully, transparently, and only for the purposes relevant to the operations of the library. For example, users' contact email address / phone numbers will only be used by the library staff to contact regarding overdue loans and fines. Personal details will be kept accurate and up to date, but will not be kept for longer than necessary. Personal details associated with library users will be deleted 90 days after the user departs from the college (e.g. student graduates, staff leaves position).

The library database system could be further developed to include the following functionalities:

- Library members to reserve and request resources.
- Library members to extend current loans.
- Library members to bookmark resources in their account.
- Library staff to remove resources from the database when the resource is out of circulation.
- Additional resource types.
- Additional types of memberships beyond staff and students.

11. APPENDIX

Coursework 2: Database Implementation

Date due: Friday December 17th 2021.

This deliverable is concerned with the implementation of your relational schema from coursework 1, using a relational database management system (DBMS(, It is strongly recommended to use either Oracle or MySQL.

You should base your implementation on the normalised relational schema developed in coursework 1 (original or refined). The coursework should be delivered in a form so that the tables can be easily created, data inserted and queries and views run easily by the person marking the work. This means that screen shots of code are not acceptable, please include the text of the code either in separate files or copy and paste the text of the code into your main report document.

You can create your tables using individual "create table" statements, but we would recommend instead that you put all of these into one script file, so that it is easier to reconstruct your entire schema should you find data or tables have been corrupted or when you wish to make changes. You can use as a template for you create tables script the "labtables" script used to create the EMP and DEPT tables for the labs.

For coursework 2, you are asked to hand in:

- The relational schema from coursework 1. If you decide to amend this, that is fine, you are not tied to the design you had for coursework 1, but you should include the relational schema you submitted for coursework 1 and explain any changes you have made.

- A listing of all the 'create table' commands that were used to set up your database. You should make use of declarative constraints to establish primary and foreign keys and perform validation checks on data to be entered.

- The sample test data. This test data should be carefully designed in order to test that your queries will work under any conditions.

- A set of 3-4 view definitions ('create view' commands), with listings of the output you obtain when you list these views.

AND the SQL CREATE VIEW command itself. Marks will be awarded for the appropriateness of the views created in the context of the library system, that is, how useful they are likely to be to individuals or groups of users of the system.

- A set of about 12 SQL queries with listings of the output you obtain when you run the query. Marks will be awarded for use of a wide range of SQL language constructs and the relevance of the query to individuals or groups of users of the library system. A query, in this context, consists of both the English explanation of what the query is intended to do, the SQL code for the query and the output it produces. Remember that screen prints of code are not acceptable, you must submit the proper text of the query code, though screen prints of the output produced by the query are acceptable.

- A set of 4 database triggers used to implement business rules or perform validation, auditing or replication processes in the system. A trigger in this context includes an English specification of what the trigger is intended to do, the code of the trigger itself and a demonstration of trigger execution.

Note that it is essential that you do actually build the database AND test out all the queries / views/triggers that you include in your report. Any evidence that this has not been done, or that any problems have not been honestly reported, will be severely penalised. In particular, any evidence that code has been included that won't actually run on the DBMS being used, unless clearly identified as such, will result in a mark of 0% for the whole coursework submission.

Finally, you should include up to a page discussing the data security issues that would need to be addressed by the administrators and users of the system, including complying with the UK Data Protection Act 2018.