College Library Management System

ECS740P - Database Systems

Coursework 2

Group 15

Haibo Li, Tsz Kin Law, Sijia Peng, Zhaoxiang Zhang

Introduction

This report is a continuation of the previous coursework and focuses on the application of a relational schema on Oracle Live SQL. Fundamental components include: creating tables and views, creating test data and creating queries to test the functionality of this library database. The goal is to build a robust database system which fulfils the requirements of Coursework 1’s design and assumptions. Below is the conceptual relational schema from the conclusion of the previous coursework. Please note that the underlined font represents the primary key and italic font represents a foreign key.

Loan_limit (<u>loan_limit_id</u>, class, loan_limit)
Member (<u>member_id</u>, personal_id, fname, lname, dob, email, mobile, billing_address, *loan_limit_id*)
Department (<u>department_id</u>, department_name)
Staff (<u>staff_id</u>, *personal_id*, *department_id*, position)
Student (<u>student_id</u>, *personal_id*, *department_id*, major)
Subject (<u>subject_id</u>, subject_name)
Map (<u>map_id</u>, floor, shelf)
Loan_type (<u>loan_type_id</u>, loan_period)
Resource (<u>resource_id</u>, <u>copy_id</u>, *map_id*, *loan_type_id*)
Resource_subject(<u>*resource_id*</u>, *subject_id*)
CD (<u>*cd_id*</u>, title, singer, manufacturer, release_date, no_of_disc)
Book (<u>*ISBN*</u>, title, author, publisher, publish_date, language, paperback)
DVD (<u>*dvd_id*</u>, title, director, main_actor_1, main_actor_2, studio, run_time)
Loan (<u>loan_id</u>, *member_id*, *resource_id*, *copy_id*, loan_date, return_date)
Fine (<u>fine_id</u>, *loan_id*, paid_date)

Relational Schema

Although we have finished constructing the conceptual design of the database application in the previous coursework, there are still some modifications and adjustments that could be fine-tuned to optimise the schema.

The ‘loan_limit’ table stores types of membership which imply how many resources a library member can borrow at one time. Therefore ‘loan_limit’ should be an attribute instead of a table name. The entity is renamed to ‘membership’ and the attribute ‘class’ is renamed to ‘member_type’, making it more semantically correct.

For ‘member’, ‘staff’ and ‘student’ tables, they all have an attribute ‘personal_id’ which is unique or referred to as a candidate key. When duplicating multiple ids across multiple tables, the schema is opened to possible data integrity issues. As a single and unique ‘member_id’ issued by the library is sufficient, we have chosen to remove ‘personal_id’ in ‘member’ and replace the foreign key in ‘staff’ and ‘student’ to ‘member_id’ as well. It still serves the functionality to distinguish parent class and subclasses and prevent a single person from registering more than one library membership. ‘member’ is renamed to ‘library_member’ for semantic accuracy.

Resource related entities could also be modified. Due to the fact the ‘resource’ table has a composite primary key and the subject is not fully dependent on the composite primary key, we h ave to separate the subject and build a new relation as ‘resource_subject’ which leads to a less concise structure. Hence, by assigning each copy of a resource a unique ‘item_id’, it prevents the problem above caused by using a composite primary key. ‘resource’ table, as a result, is renamed to ‘item_’. However, ‘item’ table could in fact have a duplicate resource_id because of the copy issue. ‘cd_id’, ‘ISBN’ and ‘dvd_id’ cannot reference ‘resource_id’ due to the rules of foreign key (it can only reference candidate/primary key which is unique). We then create a new table ‘resource’ which only has one attribute ‘resource_id’ and serves as a medium to correctly link ’book’,’dvd’ and ‘cd’ entities together with the item table. Meanwhile, as ‘subject’ is individually separated, we create a new table ‘subject_area’ to systemize subject areas as each floor can be dissected into shelves with specific subjects. We should also mention that the attribute ‘paperback’ in ‘book’ is replaced with ‘pages’, which we thought would be more useful to a library.

As resource related entities are amended, ‘loan’ now references ‘item_id’ instead of ‘resource_id’. Since the ‘loan_’ record has a one to one relationship with the ‘fine’ record, we have removed ‘fine_id’ and taken ‘loan_id’ as the primary key of the ‘fine’ table.

Lastly, we also renamed some entities to avoid reserved words in Oracle. The finalized version of the library database relational schema can be found below, which should be identical to those created by the following SQL commands.

Resource_ (resource_id)
CD (*cd_id*, title, singer, manufacturer, release_date, no_of_disc)
Book (*ISBN*, title, author, publisher, publish_date, language, paperback)
DVD (*dvd_id*, title, director, main_actor_1, main_actor_2, studio, run_time)
Subject (subject_id, subject_name)
Area (area_id, floor, shelf)
Subject_area (subject_area_id, *subject_id*, *area_id*)
Loan_type (loan_type_id, loan_period)
Item_ (item_id, *resource_id*, copy_id, *subject_area_id*, *loan_type_id*)
Membership (member_type_id, member_type, loan_limit)
Library_member (member_id, fname, lname, dob, email, mobile, billing_address, *member_type_id*)
Department (department_id, department_name)
Staff (staff_id, *member_id*, *department_id*, position)
Student (student_id, *member_id*, *department_id*, major)
Loan (loan_id, *member_id*, *item_id*, loan_date, return_date)
Fine (*loan_id*, paid_date)

Creating Tables

This section will focus on the data definition language (DDL) when constructing the schema. It includes the table creation commands and the declarative constraints imposed, followed by a short description. It is worth mentioning that the tables below follow a sequential order to ensure referential integrity required by the foreign keys.

Resource_

```
CREATE TABLE resource_ (
   resource_id VARCHAR2(13) NOT NULL,
   CONSTRAINT resource_pk PRIMARY KEY (resource_id),
   CONSTRAINT resource_check_id_format CHECK (REGEXP_LIKE (resource_id, '(^[CD]¥d+$)|(^¥d+$)')),
   CONSTRAINT resource_check_id_length CHECK (LENGTH(resource_id) BETWEEN 9 AND 13)
);
```

The resource table acts like a catalogue of all the current resources. Therefore, it only has one attribute, which is the ‘resource_id’. We declare it has a data type VARCHAR2 with a maximum length of 13 characters. It is because the maximum length of an ID is ISBN with 13 digits. ‘resource_id’ could not be null and we give it a primary key constraint and two check constraints with the help of regular expression and length function, ensuring the incoming values must be in the prescribed format: 13 / 10 digits ISBN or starting with ‘D’ / ‘C’ + 8 – 10 digits, as stated in the assumption of the previous coursework.

CD

```
CREATE TABLE CD (
   cd_id VARCHAR2(11) NOT NULL,
   title VARCHAR2(50) NOT NULL,
   singer VARCHAR2(30) NOT NULL,
   manufacturer VARCHAR2(30) NOT NULL,
   release_date DATE NOT NULL,
   no_of_disc NUMBER(2) NOT NULL,
   CONSTRAINT cd_pk PRIMARY KEY (cd_id),
   CONSTRAINT cd_fk FOREIGN KEY (cd_id) REFERENCES resource_(resource_id),
   CONSTRAINT cd_check_id_format CHECK (REGEXP_LIKE (cd_id, '(^[C]¥d+$)')),
   CONSTRAINT cd_check_id_length CHECK (LENGTH(cd_id) BETWEEN 9 AND 11)
);
```

The cd table stores all necessary details of each CD tuple. We declare ‘cd_id’ to have a data type VARCHAR2 with a maximum length of 11 digits. ‘cd_id’ could not be null and we give it a primary key constraint and a foregin key constraint. Although a similar check has been performed in the resource table and ‘cd_id’ is referencing ‘resource_id’, it includes formats of ‘dvd_id’, ‘ISBN’ and ‘cd_id’ altogether. Therefore, we have to ensure the incoming value of ‘cd_id’ fits the prescribed format by imposing two check constraints: ‘cd_check_id_format’ and ‘cd_check_id_length’ with the help of regular expression and the length function. Hence, ‘cd_id’ must start with ‘C’, plus a group of digits and the total length must be within the range of 9 to 11. For the remaining attributes, we give them VARCHAR2 or DATE data types with respect to the values’ nature.

Book

```
CREATE TABLE book (
   ISBN VARCHAR2(13) NOT NULL,
   title VARCHAR2(50) NOT NULL,
   author VARCHAR2(30) NOT NULL,
   publisher VARCHAR2(30) NOT NULL,
   publish_date DATE NOT NULL,
   lang VARCHAR2(20) NOT NULL,
   pages NUMBER(4) NOT NULL,
   CONSTRAINT book_pk PRIMARY KEY (ISBN),
   CONSTRAINT book_fk FOREIGN KEY (ISBN) REFERENCES resource_(resource_id),
   CONSTRAINT book_check_isbn_format CHECK (REGEXP_LIKE (ISBN, '(^¥d+$)')),
   CONSTRAINT book_check_isbn_length CHECK (LENGTH(ISBN) = 10 OR LENGTH(ISBN) = 13)
);
```

The book table stores all necessary details of each book tuple. We declare ‘ISBN’ to have a data type VARCHAR2 with a maximum length of 13 digits following its international standard. ‘ISBN’ could not be null and we give it a primary key constraint and a foregin key constraint. Although a similar check has been performed in the resource table and ‘ISBN’ is referencing ‘resource_id’, it includes formats of ‘dvd_id’, ‘ISBN’ and ‘cd_id’ together. Similar to’ cd_id’ we have to ensure the incoming value of ‘ISBN’ fits the prescribed format by imposing two check constraints: ‘book_check_isbn_format’ and ‘book_check_isbn_length’ with the help of regular expression and the length function. Hence, ‘ISBN’ must be a group of digits and the total length must be either 10 or 13. For the remaining attributes, we give them VARCHAR2, DATE and NUMBER data types with respect to the values’ nature.

DVD

```
CREATE TABLE DVD (
   dvd_id VARCHAR2(11) NOT NULL,
   title VARCHAR2(50) NOT NULL,
   main_actor_1 VARCHAR2(30) NOT NULL,
   main_actor_2 VARCHAR2(30) NOT NULL,
   studio VARCHAR2(30) NOT NULL,
   release_date DATE NOT NULL,
   run_time NUMBER(3) NOT NULL,
   CONSTRAINT dvd_pk PRIMARY KEY (dvd_id),
   CONSTRAINT dvd_fk FOREIGN KEY (dvd_id) REFERENCES resource_(resource_id),
   CONSTRAINT dvd_check_id_format CHECK (REGEXP_LIKE (dvd_id, '(^[D]¥d+$)')),
   CONSTRAINT dvd_check_id_length CHECK (LENGTH(dvd_id) BETWEEN 9 AND 11)
);
```

The dvd table stores all necessary details of each DVD tuple. We declare ‘dvd_id’ to have a data type VARCHAR2 with a maximum length of 11 digits following its international standard. ‘dvd_id’ could not be null and we give it a primary key constraint and a foregin key constraint. As stated previously we have to ensure the incoming value of ‘dvd_id’ must fit its prescribed format by imposing two check constraints: ‘dvd_check_id_format’ and ‘dvd_check_id_length’ with the help of regular expression and the length function. Hence, ‘dvd_id’ must start with ‘D’, plus a group of digits and the total length must be within the range of 9 to 11. For the remaining attributes, we give them VARCHAR2, DATE and NUMBER data types with respect to the values’ nature.

Subject

```
CREATE TABLE subject (
   subject_id NUMBER(2) NOT NULL,
   subject_name VARCHAR2(30) NOT NULL,
   CONSTRAINT subject_pk PRIMARY KEY (subject_id)
);
```

The subject table stores all necessary details of each subject tuple. We declare ‘subject_id’ to have a data type NUMBER with a maximum of 2 digits as we believe there would not be more than 99 subjects. It is also given a primary key constraint. ‘subject_name’ has a data type VARCHAR2(30) with NOT NULL constraint as we cannot have a subject_id which points to no name.

Area

```
CREATE TABLE area (
   area_id NUMBER(3) NOT NULL,
   floor NUMBER(1) NOT NULL,
   shelf NUMBER(2) NOT NULL,
   CONSTRAINT area_pk PRIMARY KEY (area_id),
   CONSTRAINT area_max_floor CHECK (floor BETWEEN 0 AND 2),
   CONSTRAINT area_max_shelf CHECK (shelf BETWEEN 1 AND 50)
);
```

The area table stores all necessary details of each area location tuple, which is equivalent to a specific shelf in each floor. We declare ‘area_id’ to have a data type NUMBER with a maximum of 3 digits as the total shelves are 150. ‘area_id’ is also given a primary key constraint. ‘area_max_floor’ and ‘area_max_shelf’ constraints exist as we assumed the library has three floors and each floor has a maximum of 50 shelves in the previous coursework.

Subject_area

```
CREATE TABLE subject_area (
   subject_area_id NUMBER(3) NOT NULL,
   subject_id NUMBER(2) NOT NULL,
   area_id NUMBER(3) NOT NULL,
   CONSTRAINT subject_area_pk PRIMARY KEY (subject_area_id),
   CONSTRAINT subject_area_fk1 FOREIGN KEY (subject_id) REFERENCES subject(subject_id),
   CONSTRAINT subject_area_fk2 FOREIGN KEY (area_id) REFERENCES area(area_id)
);
```

The subject area table stores all the combinations of subject and area location pairs. We declare ‘subject_area_id’ to be the primary key, ‘subject_id’ and ‘area_id’ to be foreign keys and reference the subject table and the area tables respectively through two constraints: ‘subject_area_fk1’ and ‘subject_area_fk2’.

Loan_type

```
CREATE TABLE loan_type (
   loan_type_id NUMBER(2) NOT NULL,
   loan_period NUMBER(2) NOT NULL,
   CONSTRAINT loan_type_pk PRIMARY KEY (loan_type_id),
   CONSTRAINT loan_type_check_period CHECK (loan_period = 0 OR loan_period = 2 OR loan_period = 14)
);
```

The loan type table stores all the types of loan. We declare ‘loan_type_id’ to be the primary key. As currently there are only three types of loan for an item as prescribed, we restrict the value ‘loan_period’ such that it can only be 0 or 2 or 14 by the constraint ‘loan_type_check_period’.

Item_

```
CREATE TABLE item_ (
   item_id VARCHAR2(5) NOT NULL,
   resource_id VARCHAR2(13) NOT NULL,
   copy_id NUMBER(2) NOT NULL,
   subject_area_id NUMBER(3) NOT NULL,
   loan_type_id NUMBER(2) NOT NULL,
   CONSTRAINT item_pk PRIMARY KEY (item_id),
   CONSTRAINT item_fk1 FOREIGN KEY (resource_id) REFERENCES resource_(resource_id),
   CONSTRAINT item_fk2 FOREIGN KEY (subject_area_id) REFERENCES subject_area(subject_area_id),
   CONSTRAINT item_fk3 FOREIGN KEY (loan_type_id) REFERENCES loan_type(loan_type_id),
   CONSTRAINT item_check_id_format CHECK (REGEXP_LIKE (item_id, '(^¥d+$)'))
```

);

The item_ table stores all necessary details of each copy of a resource. We declare ‘item_id’ to have a data type VARCHAR2 with a maximum length of 5 digits. ‘item_id’ is a string of numbers and as the library holds a maximum of 99999 items due to its small size, we limit the length of it. ‘item_id’ could not be null and we give it a primary key constraint and a regular expression check constraint: ‘item_check_id_format’, ensuring it is a string of digits. Foreign key constraints are imposed on ‘resource_id’, ‘subject_area_id’ and ‘loan_type_id’ to ensure data integrity. As the format checking has been done in the parent tables, we can rely on the foreign key constraint only in the child table. ‘copy_id’ is restricted to a maximum of two digits as we believe there would not be a resource to have more than 99 copies.

Membership

```
CREATE TABLE membership (
   member_type_id NUMBER(2) NOT NULL,
   member_type VARCHAR2(20) NOT NULL,
   loan_limit NUMBER(2) NOT NULL,
   CONSTRAINT membership_pk PRIMARY KEY (member_type_id)
);
```

The membership table stores all the types of members. We declare that the data type of ‘member_type_id’ is NUMBER with the maximum length of 2 digits and give it a primary key constraint. Although currently there are only two types of members: student and staff, roles could be increased in the future to satisfy members’ needs. For example, the library manager could grant PhD students the ability to borrow 7 items, which is in between the 5 and 10 of the loan limits of student and staff respectively. Loan limits can also be altered in different periods, such as midterms and finals. In order to keep the flexibility, we do not strictly impose constraints on member types and loan limits.

Library_member

```
CREATE TABLE library_member (
   member_id VARCHAR2(7) NOT NULL,
   fname VARCHAR2(30) NOT NULL,
   lname VARCHAR2(20) NOT NULL,
   dob DATE NOT NULL,
   email VARCHAR2(50) NOT NULL,
   mobile VARCHAR2(15) NOT NULL,
   billing_address VARCHAR2(100) NOT NULL,
   member_type_id NUMBER(2) NOT NULL,
   CONSTRAINT library_member_pk PRIMARY KEY (member_id),
```

```
    CONSTRAINT library_member_fk FOREIGN KEY (member_type_id) REFERENCES membership(member_type_id),
    CONSTRAINT library_member_check_id_format CHECK (REGEXP_LIKE (member_id, '(^(LIB)¥d+$)')),
    CONSTRAINT library_member_check_email CHECK (email LIKE '%@%' AND email LIKE '%.%')
);
```

The library member table stores all necessary details of each library member. We declare 'member_id' to have a data type VARCHAR2 with a maximum length of 7 digits. 'member_id' is a string of digits with 'LIB' upfront. Due to the size of the school, we expect the library to have at most 9999 members. 'member_id' could not be null and we give it a primary key constraint and a regular expression check constraint: library_member_check_id_format', ensuring it fits within the format as mentioned above. Foreign key constraints are imposed on 'member_type_id' to ensure data integrity. We also impose a check constraint: 'library_member_check_email' on the email entry to make sure it aligns with the correct format of an email address. For the remaining attributes, we give them VARCHAR2, DATE and NUMBER data types with respect to the values' nature.

Department

```
CREATE TABLE department (
    department_id NUMBER(2) NOT NULL,
    department_name VARCHAR2(30) NOT NULL,
    CONSTRAINT department_pk PRIMARY KEY (department_id)
);
```

The department table stores all the minimum necessary information of each department for this library database. We declare 'department_id' to be a primary key with a data type NUMBER(2), implying there could not be more than 99 departments.

Staff

```
CREATE TABLE staff (
    staff_id VARCHAR2(4) NOT NULL,
    member_id VARCHAR2(7) NOT NULL,
    department_id NUMBER(2) NOT NULL,
    position VARCHAR2(20) NOT NULL,
    CONSTRAINT staff_pk PRIMARY KEY (staff_id),
    CONSTRAINT staff_fk1 FOREIGN KEY (member_id) REFERENCES library_member(member_id),
    CONSTRAINT staff_fk2 FOREIGN KEY (department_id) REFERENCES department(department_id),
    CONSTRAINT staff_check_id_format CHECK (REGEXP_LIKE (staff_id, '(^¥d+$)'))
);
```

The staff table stores all the necessary information for each staff member. We declare that the data type of 'staff_id' is VARCHAR2 with a maximum length of 4. We impose a primary key constraint and a regular expression check constraint: 'staff_check_id_format' on 'staff_id' to ensure it is a string of digits. Foreign key restrictions are declared to guarantee "member_id" and "department_id" are references.

Student

```
CREATE TABLE student (
   student_id VARCHAR2(4) NOT NULL,
   member_id VARCHAR2(7) NOT NULL,
   department_id NUMBER(2) NOT NULL,
   major VARCHAR2(40) NOT NULL,
   CONSTRAINT student_pk PRIMARY KEY (student_id),
   CONSTRAINT student_fk1 FOREIGN KEY (member_id) REFERENCES library_member(member_id),
   CONSTRAINT student_fk2 FOREIGN KEY (department_id) REFERENCES department(department_id),
   CONSTRAINT student_check_id_format CHECK (REGEXP_LIKE (student_id, '(^¥d+$)'))
);
```

The student table stores all the necessary information for each student member. We declare that the data type of 'student_id' is VARCHAR2 with a maximum length of 4. We impose a primary key constraint and a regular expression check constraint: 'student_check_id_format' on 'student_id' to ensure it is a string of digits. Foreign key restrictions are declared to guarantee "member_id" and "department_id" are references.

Loan

```
CREATE TABLE loan (
   loan_id NUMBER NOT NULL,
   member_id VARCHAR2(7) NOT NULL,
   item_id VARCHAR2(5) NOT NULL,
   loan_date DATE NOT NULL,
   return_date DATE,
   CONSTRAINT loan_pk PRIMARY KEY (loan_id),
   CONSTRAINT loan_fk1 FOREIGN KEY (member_id) REFERENCES library_member(member_id),
   CONSTRAINT loan_fk2 FOREIGN KEY (item_id) REFERENCES item_(item_id)
);
```

The loan table stores all the necessary information for each loan record. We declare that the 'loan_id' is the primary key and is data type NUMBER with no maximum digits restriction as it is hard to predict how many loans there will be when the loan table also records historical loans. Foreign key restrictions are declared to guarantee 'member_id' and 'item_id' are references. 'return_date' is specifically set to nullable as the system will fill up the date value once the loanee returns the loan item.

Fine

```
CREATE TABLE fine (
```

```
   loan_id NUMBER NOT NULL,
   paid_date DATE,
   CONSTRAINT fine_pk PRIMARY KEY (loan_id),
   CONSTRAINT fine_fk FOREIGN KEY (loan_id) REFERENCES loan(loan_id)
);
```

The loan table stores all the necessary information for each fine record. We declare that the ‘loan_id’ is the primary key in data type NUMBER with no maximum digits restriction as it is in the loan table. Foreign key restrictions are declared to guarantee ‘loan_id’ is a reference. ‘fine_date’ is specifically set to nullable as the system will fill up the date value once the loanee pays the fine.

Creating Data

This section will focus on the first part of insert in the Data Manipulation Language (DML) when constructing the schema. After creating the table, we also need to insert the corresponding sample data in each table to test the overall integrity of the library database application. We will demonstrate a sample insert statement for each table, followed by a query search to display all the sample data. For those tables with additional declarative constraints we have also included a sample insert statement which triggers the declarative constraint if you wish to test them.

membership

INSERT INTO membership VALUES(01,'staff', 10);

.

.

.

SELECT * FROM membership;

MEMBER_TYPE_ID	MEMBER_TYPE	LOAN_LIMIT
1	staff	10
2	student	5

library_member

INSERT INTO library_member VALUES

('LIB001','Kennith','Barber',TO_DATE('27-03-1956', 'DD-MM-YYYY'),'ken_barber@microsoft.com','07665189526','18 Church Road, Ton Pentre, CF41 7ED',01);

.

.

.

SELECT * FROM library_member;

MEMBER_ID	FNAME	LNAME	DOB	EMAIL	MOBILE	BILLING_ADDRESS	MEMBER_TYPE_ID
LIB001	Kennith	Barber	27-MAR-56	ken_barber@microsoft.com	07665189526	18 Church Road, Ton Pentre, CF41 7ED	1
LIB002	Kennith	Jacket	21-OCT-00	kenny_jacket_66@gmail.com	07651465179	15 Montgomery Way, Chandler Ford, SO53 3PX	2
LIB003	Brad	Pitman	15-SEP-62	bradpitters999@gmail.com	07186848981	4 Comma Close, Newcastle Upon Tyne, NE13 9EE	1
LIB004	Glen	Johnson	28-JUN-00	glenjohnsonpfc@gmail.com	07921882615	227 Lancaster Road North, Preston, PR1 2PY	2
LIB005	Yan	Freund	15-SEP-62	yfreund@yahoo.ca	07156517981	Sawpit House, High Street, Hogsthorpe, PE24	1

						5ND	
LIB006	Stacy	Takasa	15-SEP-98	stakasa321@gmail.com	07185529294	56 Dennis Road, East Molesey, KT8 9ED	2
LIB007	Mark	Feamerson	01-FEB-77	mark_feamster31@comcast.net	07399511822	34 Farm Lane, Plymouth, PL5 3PQ	1
LIB008	Winston	Hernandez	19-AUG-94	wapoyar852@hide-mail.net	07198481668	32 Coppice Gate, Cheltenham, GL51 9QL	2
LIB009	Rebecca	Barber	21-SEP-80	bakerrebecca@hotmail.com	07198529848	3 Chellow Dene, Mossley, OL5 0NB	1
LIB010	Fergus	McCalister	01-JUL-97	perezbrian@bryant.org	07195294941	2 Eaton Gardens, Spalding, PE11 1GY	2
LIB011	Christopher	White	27-MAR-91	christopherwhite@perez.biz	07189829628	2 Thornycroft Avenue, Southampton, SO19 9EA	1
LIB012	Amanda	Arnold	05-MAY-91	amanda23@hotmail.com	07189562981	17 Westfield Crescent, Thurnscoe, S63 0PU	2
LIB013	Rebecca	Rodriguez	09-APR-93	rebecca63@jimenez.biz	07298519818	123 High Street, Iver, SL0 9QB	1
LIB014	Melissa	Reeves	18-NOV-99	melissa14@gmail.com	07289819818	26 Vicarage Road, Hastings, TN34 3NA	2

Below we have included some insert statements which trigger our integrity constraints to show that they are functioning:

INSERT INTO library_member VALUES ('001','Kennith','Barber',TO_DATE('27-03-1956', 'DD-MM-YYYY'),'ken_barber@microsoft.com','07665189526','18 Church Road, Ton Pentre, CF41 7ED',01);

-- member_id must be a string starting with ' LIB' and the maximum length of it is 7 digits.

INSERT INTO library_member VALUES ('LIB001','Kennith','Barber',TO_DATE('27-03-1956', 'DD-MM-YYYY'), 'ken_barbermicrosoft.com','07665189526','18 Church Road, Ton Pentre, CF41 7ED',01);

-- the email must be like 'XX@XX' , but in this statement, the '@' is missing.

department

INSERT INTO department VALUES (01,'computer science');

.

.

.

SELECT * FROM department;

DEPARTMENT_ID	DEPARTMENT_NAME
1	computer science
2	electronic engineering

staff

INSERT INTO staff VALUES ('001', 'LIB001', 01, 'professor');

.

.

.

SELECT * FROM staff;

STAFF_ID	MEMBER_ID	DEPARTMENT_ID	POSITION
001	LIB001	1	professor
002	LIB003	2	professor
003	LIB005	1	lecturer
004	LIB007	2	lecturer
005	LIB009	1	teaching assistant
006	LIB011	1	teaching assistant
007	LIB013	2	teaching assistant

Below we have included an insert statement to trigger our integrity constraints to show that they are functioning:

INSERT INTO staff VALUES ('s01', 'LIB001', 01, 'professor');

--staff_id must be a string of numbers.

student

INSERT INTO student VALUES ('001', 'LIB002', 01, 'computer science');

.

.

.

SELECT * FROM student;

STUDENT_ID	MEMBER_ID	DEPARTMENT_ID	MAJOR
001	LIB002	1	computer science
002	LIB004	1	computer science
003	LIB006	1	computing and information systems
004	LIB008	1	big data science
005	LIB010	2	electronic engineering
006	LIB012	2	electronic engineering
007	LIB014	2	telecommunication and wireless systems

Below we have included an insert statement to trigger our integrity constraints to show that they are functioning:

```
INSERT INTO student VALUES ('st01', 'LIB002', 01, 'computer science');
--student_id must be a string of numbers.
```

resource_

```
INSERT INTO resource_ (resource_id) VALUES ('D1137420190');
.
.
.
SELECT * FROM resource_;
```

RESOURCE_ID
0008328927
0385093799
9780007525522
9780152465049
9780345339737
9781338299151
9781786751041
C1183748328
C1183758443
C40281531
C55112945
C607207102
C732952558
C988066191
D1018237377
D1039921794
D1137420190
D122347647
D1237831462
D180159984
D999576545

Below we have included an insert statement to trigger our integrity constraints to show that they are functioning:

INSERT INTO resource_ (resource_id) VALUES ('f2137420190');
--resource_id must be 13 / 10 digits ISBN or starting with 'D' / 'C' + 8 - 10 digits

book

INSERT INTO book VALUES ('9780345339737', 'The Lord of the Rings', 'J.R.R. Tolkien', 'George Allen & Unwin', DATE '1955-07-29', 'English', 416);

.

.

.

SELECT * FROM book;

ISBN	TITLE	AUTHOR	PUBLISHER	PUBLISH_DATE	LANG	PAGES
9780345339737	The Lord of the Rings	J.R.R. Tolkien	George Allen & Unwin	29-JUL-55	English	416
9780152465049	The Little Prince	Antoine de Saint-Exupéry	Petroleum Industry Press	01-APR-43	French	96
9781338299151	Harry Potter and the Chamber of Secrets	J.K. Rowling	Scholastic Paperbacks Express	02-JUL-98	English	251
9781786751041	Alices Adventures in Wonderland	Lewis Carroll	Little Simon Express	01-JAN-65	English	352
9780007525522	The Hobbit	John Ronald Riel Tolkien	Harpercollins Express	21-SEP-37	English	310
0008328927	And Then There Were None	Agatha Christie	Daily Express	06-NOV-39	English	272
0385093799	Dream of the Red Chamber	Xueqin Cao	Writes Express	01-JUL-53	Chinese	352

Below we have included an insert statement to trigger our integrity constraints to show that they are functioning:
INSERT INTO book VALUES ('8328927', 'And Then There Were None', 'Agatha Christie', 'Daily Express', DATE '1939-11-06', 'English', 272);
-- 'ISBN' must be a group of digits and the total length must be either 10 or 13

CD

INSERT INTO CD VALUES ('C988066191', 'Coast to Coast', 'Westlife', 'BMG',TO_DATE('2000-11-06', 'YYYY-MM-DD'), 2);

.

.

.

SELECT * FROM cd;

CD_ID	TITLE	SINGER	MANUFACTURER	RELEASE_DATE	NO_OF_DISC
C988066191	Coast to Coast	Westlife	BMG	06-NOV-00	2
C607207102	Meteora	Linkin Park	Warner Records	25-MAR-03	3
C40281531	Enrique Iglesias	Enrique Iglesias	Warner Records	12-JUL-95	1
C55112945	Sanctuary	Simon Webbe	BMG	14-NOV-05	1
C732952558	And Rising	98 degrees	Universal Music	12-MAR-02	3
C1183758443	The Platinum Collection	Blue	Warner Records	12-SEP-03	4
C1183748328	The Best of Ricky Martin	Ricky Martin	Sony Bmg Europe	08-JAN-08	3

Below we have included an insert statement to trigger our integrity constraints to show that they are functioning:

INSERT INTO CD VALUES ('q988066191', 'Coast to Coast', 'Westlife', 'BMG',TO_DATE('2000-11-06', 'YYYY-MM-DD'), 2);

--cd_id must start with 'C', plus a group of digits and the total length must be within the range of 9 to 11.

DVD

INSERT INTO DVD VALUES ('D999576545', 'The Pursuit of Happyness', 'Will Smith', 'Jaden Smith', 'Relativity Media', DATE '2006-12-15', 117);

.

.

.

SELECT * FROM dvd;

DVD_ID	TITLE	MAIN_ACTOR_1	MAIN_ACTOR_2	STUDIO	RELEASE_DATE	RUN_TIME
D1137420190	The Shawshank Redemption	Tim Robbins	Morgan Freeman	Castle Rock Entertainment	23-SEP-94	142
D122347647	Forrest Gump	Tom Hanks	Robin White	Paramount	04-OCT-94	142
D999576545	The Pursuit of Happyness	Will Smith	Jaden Smith	Relativity Media	15-DEC-06	117

D180159984	Braveheart	Mel Gibson	Kathleen McCome	Icon Production	24-MAY-95	177
D1237831462	A Beautiful Mind	Russell Crow	Jennifer Connelly	Universal Studios	21-DEC-01	135
D1039921794	Million Dollar Baby	Clint Eastwood	Hilary Swank	Warner Bros. Entertainment	15-DEC-04	132
D1018237377	Halt and Catch Fire	Lee Pace	Scott McNellie	AMC Studios	01-JUN-04	53

Below we have included an insert statement to trigger our integrity constraints to show that they are functioning:

INSERT INTO DVD VALUES ('122347647', 'Forrest Gump', 'Tom Hanks', 'Robin White', 'Paramount', DATE '1994-10-04', 142);

-- 'dvd_id' must start with 'D', plus a group of digits and the total length must be within the range of 9 to 11

loan_type

INSERT INTO loan_type (loan_type_id, loan_period)VALUES(1, 2);

.

.

.

SELECT * FROM loan_type;

LOAN_TYPE_ID	LOAN_PERIOD
1	2
2	14
3	0

Below we have included an insert statement to trigger our integrity constraints to show that they are functioning:

INSERT INTO loan_type (loan_type_id, loan_period) VALUES (3, 6);

--We assume we only have three loan_period, 0,2,14.

subject

INSERT INTO subject VALUES (1, 'English Literature');

.

.

.

SELECT * FROM subject;

SUBJECT_ID	SUBJECT_NAME
1	English Literature

2	Chinese Literature
3	Pop Music
4	Comedy Movies
5	Drama Movies
6	Action Movies
7	Drama TV Series

area

INSERT INTO area VALUES(1, 0, 1);

.

.

.

SELECT * FROM area;

AREA_ID	FLOOR	SHELF
1	0	1
2	0	2
3	0	3
4	1	4
5	1	5
6	1	7
7	2	1

Below we have included an insert statement to trigger our integrity constraints to show that they are functioning:

INSERT INTO area (area_id, floor, shelf) VALUES (1, 3, 1);

--we assume we only have 3 floors and each floor have 50 shelves, so ‘area_max_floor’ and ‘area_max_shelf’ must be between 0-2 and 0-50.

subject_area

INSERT INTO subject_area (subject_area_id, subject_id, area_id)VALUES(1, 1, 1);

.

.

.

SELECT * FROM subject_area;

SUBJECT_AREA_ID	SUBJECT_ID	AREA_ID
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7

item_

INSERT INTO item_ VALUES(1000,'D1137420190',1,5,1);

.

.

.

SELECT * FROM item_;

ITEM_ID	RESOURCE_ID	COPY_ID	SUBJECT_AREA_ID	LOAN_TYPE_ID
1000	D1137420190	1	5	1
1001	D1137420190	2	5	1
1002	D1137420190	3	5	2
1003	D122347647	1	4	1
1004	D122347647	2	4	1
1005	D122347647	3	4	2
1006	D122347647	4	4	2
1007	D122347647	5	4	2
1008	D999576545	1	5	1
1009	D999576545	2	5	2
1010	D180159984	1	6	1
1011	D180159984	2	6	1
1012	D180159984	3	6	1
1013	D180159984	4	6	2
1014	D1237831462	1	5	1

1015	D1237831462	2	5	1
1016	D1039921794	1	5	2
1017	D1039921794	2	5	2
1018	D1018237377	1	7	1
1019	D1018237377	2	7	2
1020	D1018237377	3	7	2
1100	C988066191	1	3	1
1101	C988066191	2	3	1
1102	C988066191	3	3	1
1103	C607207102	1	3	3
1104	C607207102	2	3	3
1105	C607207102	3	3	3
1106	C40281531	1	3	2
1107	C40281531	2	3	2
1108	C40281531	3	3	2
1109	C55112945	1	3	2
1110	C55112945	2	3	2
1111	C55112945	3	3	2
1112	C732952558	1	3	1
1113	C732952558	2	3	1
1114	C732952558	3	3	1
1115	C1183758443	1	3	2
1116	C1183758443	2	3	2
1117	C1183758443	3	3	2
1118	C1183748328	4	3	2
1119	C1183748328	1	3	3
1120	C1183748328	2	3	3
1121	C1183748328	3	3	3
1122	C1183748328	4	3	3
1200	9780345339737	1	1	1
1201	9780345339737	2	1	1
1202	9780345339737	3	1	1

1203	9780152465049	1	1	2
1204	9780152465049	2	1	2
1205	9780152465049	3	1	2
1206	9780152465049	4	1	2
1207	9781338299151	1	1	1
1208	9781338299151	2	1	1
1209	9781338299151	3	1	1
1210	9781338299151	4	1	3
1211	9781786751041	1	1	2
1212	9781786751041	2	1	2
1213	9781786751041	3	1	3
1214	9781786751041	4	1	3
1215	9780007525522	1	1	2
1216	9780007525522	2	1	2
1217	9780007525522	3	1	2
1218	0008328927	1	1	1
1219	0008328927	2	1	1
1220	0008328927	3	1	1
1221	0385093799	1	2	2
1222	0385093799	2	2	2
1223	0385093799	3	2	2

Below we have included an insert statement to trigger our integrity constraints to show that they are functioning:

```
INSERT INTO item_ VALUES ('it','D1137420190',1,5,1);
--item_id should be a string of digits with a maximum length of 5 digits.
```

loan

```
INSERT INTO loan VALUES(1,'LIB002',1101,TO_DATE('15-11-2021', 'DD-MM-YYYY'),TO_DATE('21-11-2021', 'DD-MM-YYYY'));
.
.
.
SELECT * FROM loan;
```

LOAN_ID	MEMBER_ID	ITEM_ID	LOAN_DATE	RETURN_DATE
1	LIB002	1101	15-NOV-21	21-NOV-21
2	LIB003	1205	12-NOV-21	15-NOV-21
3	LIB008	1104	05-DEC-21	-
4	LIB010	1109	01-NOV-21	21-NOV-21
5	LIB002	1202	23-NOV-21	29-NOV-21
6	LIB002	1209	07-DEC-21	-
7	LIB011	1002	25-NOV-21	30-NOV-21
8	LIB004	1204	25-OCT-21	-
9	LIB004	1003	25-NOV-21	11-DEC-21
10	LIB005	1200	10-DEC-21	-
11	LIB011	1201	01-DEC-21	-
12	LIB007	1202	15-DEC-21	-
13	LIB002	1106	10-DEC-21	12-DEC-21
14	LIB010	1209	15-NOV-21	23-NOV-21
15	LIB011	1107	15-OCT-21	20-OCT-21
16	LIB007	1002	15-NOV-21	19-NOV-21

fine

INSERT INTO fine VALUES(4, TO_DATE ('21-11-2021', 'DD-MM-YYYY'));

.

.

.

SELECT * FROM fine;

LOAN_ID	PAID_DATE
4	21-NOV-21
8	-
9	13-DEC-21
10	-

Creating Views

In order to enhance the usability of the database system, a set of views are created through joining specific sets of tables. In this way, users could have a broader picture of a particular field and as we derive common queries based on the views, users could also filter information and narrow the output easily. It also allows us to enforce database security as library members are only allowed to access data they require from the views.

current_avaliable_resources_locations

```
CREATE OR REPLACE VIEW current_available_resources_locations AS
(SELECT i.item_id, i.resource_id, i.copy_id, s.subject_name, a.floor, a.shelf, c.title AS cd_title,
c.singer, c.manufacturer, c.release_date AS cd_release_date, c.no_of_disc, b.title AS book_title, b
.author, b.publisher, b.publish_date, b.lang, b.pages, d.title AS dvd_title, d.main_actor_1, d.main
_actor_2, d.studio, d.release_date AS dvd_release_date, d.run_time
 FROM item_ i
  JOIN subject_area sa ON i.subject_area_id = sa.subject_area_id
   JOIN subject s ON sa.subject_id = s.subject_id
    JOIN area a ON sa.area_id = a.area_id
     FULL OUTER JOIN book b ON i.resource_id = b.isbn
      FULL OUTER JOIN cd c ON i.resource_id = c.cd_id
       FULL OUTER JOIN dvd d ON i.resource_id = d.dvd_id
        WHERE i.item_id NOT IN (SELECT item_id FROM loan WHERE return_date IS NULL));
```

This view aims to provide a macro view of resource entities related information. It is a composite of 'item_', 'subject_area', 'subject', 'loan_type', 'area', 'cd', 'book' and 'dvd'. By grouping all the aforementioned tables, when users wish to query for the location, subject, loan period or detailed description of a particular resource copy, this handy view serves as a master list and allows filtering of the output with ease.

loan_records

```
CREATE OR REPLACE VIEW loan_records AS
(SELECT l.loan_id, l.item_id, i.resource_id, i.copy_id, l.loan_date, l.return_date, lt.loan_period,
s.subject_name, lm.member_id, lm.fname, lm.lname,
ss.student_id, ss.major, ss.staff_id, ss.position, ss.department_name
 FROM loan l
  JOIN item_ i ON l.item_id = i.item_id
   JOIN loan_type lt ON i.loan_type_id = lt.loan_type_id
    JOIN subject_area sa ON i.subject_area_id = sa.subject_area_id
     JOIN subject s ON sa.subject_id = s.subject_id
      JOIN library_member lm ON l.member_id = lm.member_id
       JOIN (SELECT * FROM (
            SELECT st.student_id, st.major, sf.staff_id, sf.position,
             CASE
              WHEN st.member_id IS NULL THEN sf.member_id
              ELSE st.member_id
             END member_id,
             CASE
              WHEN st.department_id IS NULL THEN sf.department_id
```

```
          ELSE st.department_id
         END dept_id
          FROM student st
          FULL OUTER JOIN staff sf ON st.member_id = sf.member_id) m
          JOIN department d
            ON d.department_id = m.dept_id) ss ON lm.member_id = ss.member_id);
```

This view aims to provide a macro view of loans related information. It is a composite of ‘loan’, ‘item_’, ‘loan_type’, ‘subject_area’, ‘subject’, ‘library_member’, ‘student’, ‘staff’ and ‘department’ tables. By grouping all the aforementioned tables, it becomes a master list of loan records. Users can distinguish current loans between loan histories via filtering of null return dates, and calculate overdues by comparing the sum of loan_date and loan_period with system date. The member information enables calculation of loan distribution based on department, student major or staff position.

fine_records

```
CREATE OR REPLACE VIEW fine_records AS
(SELECT f.loan_id, lm.member_id, lm.fname, lm.lname, lm.dob, lm.email, lm.mobile, lm.billing_address, l.item_id, l.loan_date, lt.loan_period, l.return_date, f.paid_date,
CASE
 WHEN l.return_date IS NULL THEN '$' || FLOOR(SYSDATE - (l.loan_date + lt.loan_period))
 ELSE '$' || FLOOR(l.return_date - (l.loan_date + lt.loan_period))
END fine_amount
 FROM fine f
  JOIN loan l ON f.loan_id = l.loan_id
   JOIN item_ i ON l.item_id = i.item_id
    JOIN loan_type lt ON i.loan_type_id = lt.loan_type_id
     JOIN library_member lm ON l.member_id = lm.member_id);
```

This view aims to provide a macro view of fines related information. It is a composite of ‘fine’, ‘loan’, ‘item_’, ‘loan_type’ and ‘library_member’. By grouping all the aforementioned tables, it becomes a master list of fine records. Users can distinguish current fines between fine histories via filtering null paid dates, and check the fine amount of each record which is calculated by comparing the sum of loan_date and loan_period with system date or return date, depending on whether the resource has been returned. The financial office could also use the member contact details when collecting fines.

member_records

```
CREATE OR REPLACE VIEW member_records AS
 SELECT lm.member_id, lm.fname, lm.lname, lm.email, lm.mobile, lm.billing_address,
  ss.student_id, ss.major, ss.staff_id, ss.position, ss.department_name,
   m.loan_limit,
    COUNT(CASE WHEN l.return_date IS NULL THEN 1 END) current_borrowings, COUNT(CASE WHEN l.return_date LIKE '%' THEN 1 END) historical_borrowings
     FROM library_member lm
      JOIN membership m ON lm.member_type_id = m.member_type_id
       JOIN loan l ON lm.member_id=l.member_id
```

```
        JOIN (SELECT * FROM (
        SELECT st.student_id, st.major, sf.staff_id, sf.position,
         CASE
            WHEN st.member_id IS NULL THEN sf.member_id
            ELSE st.member_id
         END member_id,
         CASE
            WHEN st.department_id IS NULL THEN sf.department_id
            ELSE st.department_id
         END dept_id
        FROM student st
         FULL OUTER JOIN staff sf ON st.member_id = sf.member_id) m
          JOIN department d
           ON d.department_id = m.dept_id) ss ON lm.member_id = ss.member_id
            GROUP BY lm.member_id, lm.fname, lm.lname, lm.email, lm.mobile, lm.billing_address, m.loan_limit, ss.student_id, ss.major, ss.staff_id, ss.position, ss.department_name ORDER BY lm.member_id;
```

This view aims to provide a macro view of member related information. It is a composite of 'library_member', 'membership', 'loan', 'student', 'staff' and 'department'. By grouping all the aforementioned tables, it becomes a dashboard of showing library members. Users can see his or her personal information, loan limit, and the sum of current borrowings and that of historical borrowings.

Creating Queries

This section contains a list of 12 queries which we believe would be most frequently used by library members. They represent the most common queries to the library database as the output of those commands are extracting the most-wanted information by staff or members. We may not print out the complete query results as they could be lengthy, as such only the first few tuples will be displayed.

This part provides many query statements to query some specific information in the database. In this process, we can test whether the database function is normal. Below is the query statement we used and the corresponding output.

(Please separate executing SQL scripts for creating tables, data, views and triggers from the queries. LiveSQL returns no response when all commands are run at once, but is able to run when separated like this.)

1) Show list of all suspended members

This query can show members who have been suspended due to a fine of more than $10. This is very important in the library database because the suspended member cannot continue to loan books. According to this query command, we can query the basic information of these members and be suspended because of which loan.

```
SELECT od.loan_id, od.item_id, od.resource_id, od.loan_date, od.loan_period, FLOOR(SYSDATE - od.loan_date - od.loan_period) AS overdue_days,
    lm.member_id, lm.fname || lm.lname AS name, lm.email, lm.mobile
        FROM (
            SELECT loan_id, item_id, resource_id, loan_date, loan_period, member_id
                FROM loan_records WHERE return_date IS NULL AND SYSDATE - loan_date - loan_period > 10) od
                    JOIN library_member lm ON od.member_id = lm.member_id;
```

LOAN_ID	ITEM_ID	RESOURCE_ID	LOAN_DATE	LOAN_PERIOD	OVERDUE_DAYS	MEMBER_ID	NAME	EMAIL	MOBILE
11	1201	9780345339737	01-DEC-21	2	14	LIB011	ChristopherWhite	christopherwhite@perez.biz	07189829628
8	1204	9780152465049	25-OCT-21	14	39	LIB004	GlenJohnson	glenjohnsonpfc@gmail.com	07921882615
3	1104	C607207102	05-DEC-21	0	12	LIB008	WinstonHernandez	wapoyar852@hide-mail.net	0719848166

2) Show list of all resources which are being borrowed

This query statement can find all the resources that have been loaned. This can show the titles of all loaned resources and the date they were loaned. This will help library staff to know which items are not currently available.

```
SELECT l.loan_id, l.loan_date, b.title, c.title, d.title
 FROM loan l
  FULL JOIN item_ i ON l.item_id = i.item_id
   FULL JOIN book b ON i.resource_id = b.ISBN
    FULL JOIN cd c ON c.cd_id = i.resource_id
     FULL JOIN dvd d ON d.dvd_id = i.resource_id
      WHERE l.loan_id is not NULL;
```

LOAN_ID	LOAN_DATE	TITLE	TITLE	TITLE
7	25-NOV-21	-	-	The Shawshank Redemption
16	15-NOV-21	-	-	The Shawshank Redemption
9	25-NOV-21	-	-	Forrest Gump
3	05-DEC-21	-	Meteora	-
6	07-DEC-21	Harry Potter and the Chamber of Secrets	-	-
14	15-NOV-21	Harry Potter and the Chamber of Secrets	-	-
1	15-NOV-21	-	Coast to Coast	-
10	10-DEC-21	The Lord of the Rings	-	-
11	01-DEC-21	The Lord of the Rings	-	-
5	23-NOV-21	The Lord of the Rings	-	-
12	15-DEC-21	The Lord of the Rings	-	-
8	25-OCT-21	The Little Prince	-	-
2	12-NOV-21	The Little Prince	-	-
13	10-DEC-21	-	Enrique Iglesias	-
15	15-OCT-21	-	Enrique Iglesias	-
4	01-NOV-21	-	Sanctuary	-

3) Show list of all resources which are overdue

This query statement can display titles of the resources which are overdue. Also, we can see which loan and the loan date in this statement. Staff of the library can use this query to find out which items are overdue and can do further things like contacting the members who loan this resource.

```
SELECT l.loan_id, l.loan_date, b.title AS book_title, c.title AS CD_title, d.title AS DVD_title, i.copy_id
FROM loan l
FULL JOIN item_ i ON l.item_id = i.item_id
FULL JOIN book b ON i.resource_id = b.ISBN
FULL JOIN cd c ON c.cd_id = i.resource_id
FULL JOIN dvd d ON d.dvd_id = i.resource_id
JOIN loan_type lt ON i.loan_type_id = lt.loan_type_id
WHERE l.loan_id IS NOT NULL AND (SYSDATE > l.loan_date + lt.loan_period) AND l.return_date is NULL;
```

LOAN_ID	LOAN_DATE	BOOK_TITLE	CD_TITLE	DVD_TITLE	COPY_ID
3	05-DEC-21	-	Meteora	-	2
6	07-DEC-21	Harry Potter and the Chamber of Secrets	-	-	3
10	10-DEC-21	The Lord of the Rings	-	-	1
11	01-DEC-21	The Lord of the Rings	-	-	2
12	15-DEC-21	The Lord of the Rings	-	-	3
8	25-OCT-21	The Little Prince	-	-	2

4) Show most popular 5 resources

This query statement can display the five most borrowed resources and show the resource_id of these resources an d the number of how many times the resource is loaned.

```
SELECT resource_id, count(resource_id) AS counts
 FROM loan_records
  WHERE return_date IS NOT NULL
   GROUP BY resource_id
    ORDER BY counts DESC OFFSET 0 ROWS FETCH NEXT 5 ROW ONLY;
```

RESOURCE_ID	COUNTS
C40281531	2
D1137420190	2
9780345339737	1
9780152465049	1
9781338299151	1

5) Show most popular subject

This query statement can display the subject of the most popular borrowed resources, and display the name of the s ubject and the number of loans of these resources. This can help library staff to find out which subject is most popul ar and may purchase more resources for this subject.

```
SELECT subject_name, COUNT(subject_name) AS counts
 FROM loan_records
  WHERE return_date IS NOT NULL
   GROUP BY subject_name
    ORDER BY counts DESC FETCH NEXT 1 ROW ONLY;
```

SUBJECT_NAME	COUNTS
Pop Music	4

6) Show most popular resources per department

This query statement can display the most popular resources of different departments, and show the name of the departments, resource_id, and the number of loans of these resources. When knowing most departments, the library staff can purchase more copies of these resources and even analyze why students like this resource.

```
SELECT * FROM
    (SELECT department_name, resource_id, COUNT(resource_id) AS counts
    FROM loan_records GROUP BY department_name, resource_id) m
        JOIN (SELECT department_name, MAX(counts) AS max_count FROM
            (SELECT department_name, resource_id, COUNT(resource_id) AS counts
            FROM loan_records GROUP BY department_name, resource_id)
                GROUP BY department_name) s
                    ON m.department_name = s.department_name
                        WHERE m.counts = s.max_count;
```

DEPARTMENT_NAME	RESOURCE_ID	COUNTS	DEPARTMENT_NAME	MAX_COUNT
electronic engineering	D1137420190	1	electronic engineering	1
electronic engineering	9780345339737	1	electronic engineering	1
electronic engineering	C55112945	1	electronic engineering	1
electronic engineering	9780152465049	1	electronic engineering	1
electronic engineering	9781338299151	1	electronic engineering	1
computer science	9780345339737	3	computer science	3

7) Show which resources have been completely borrowed (no copies left)

This query statement can display resources that are not in stock because these resources are all borrowed, and the resource_id of the resource is returned. Library staff can consider buying more copies of these resources.

```
SELECT i.resource_id
 FROM item_ i
  JOIN loan l ON i.item_id=l.item_id
   WHERE l.return_date IS NULL
    GROUP BY i.resource_id
     HAVING COUNT (i.resource_id)-MAX(i.copy_id)=0;
```

DEPARTMENT_NAME	RESOURCE_ID	COUNTS	DEPARTMENT_NAME	MAX_COUNT
electronic engineering	D1137420190	1	electronic engineering	1
electronic engineering	9780345339737	1	electronic engineering	1
electronic engineering	C55112945	1	electronic engineering	1
electronic engineering	9780152465049	1	electronic engineering	1
electronic engineering	9781338299151	1	electronic engineerin	1

			g	
computer science	9780345339737	3	computer science	3

8) Show the location of a specific copy of a resource

This query statement can display the specific location of each copy of each resource, resource_id and copy_id will be returned to identify the specific item, and floor and shelf will be the specific location. This will help staff and member s to find the exact location of the item they want to find.

```
SELECT i.resource_id, i.copy_id, a.floor, a.shelf
 FROM area a
  JOIN subject_area sa ON sa.area_id=a.area_id
   JOIN item_ i ON i.subject_area_id=sa.subject_area_id;
```

RESOURCE_ID	COPY_ID	FLOOR	SHELF
D1137420190	1	1	5
D1137420190	2	1	5
D1137420190	3	1	5
D122347647	1	1	4
D122347647	2	1	4
D122347647	3	1	4
D122347647	4	1	4
D122347647	5	1	4
D999576545	1	1	5
D999576545	2	1	5

(we have only displayed the first 10 of 50 results)

9) Show the top 5 people who have borrowed the most resources

This query statement can display the top five members who have borrowed most in the library and will return their member_id and the specific number of borrowed items.

```
SELECT * from (SELECT member_id,COUNT(member_id) AS NUM
 FROM loan
   GROUP BY member_id
    ORDER BY COUNT(member_id) DESC ) WHERE rownum <=5;
```

MEMBER_ID	NUM
LIB002	4
LIB011	3

LIB004	2
LIB010	2
LIB007	2

10) Show where the subjects are located

This query statement can display all the subjects and where the resources of these subjects are and will return subje ct_name, floor and shelf. Library staff and members can use it to locate the subject they want to find.

```
SELECT s.subject_name, a.floor, a.shelf
FROM area a
JOIN subject_area sa ON sa.area_id=a.area_id
JOIN subject s ON s.subject_id = sa.subject_id;
```

SUBJECT_NAME	FLOOR	SHELF
English Literature	0	1
Chinese Literature	0	2
Pop Music	0	3
Comedy Movies	1	4
Drama Movies	1	5
Action Movies	1	7
Drama TV Series	2	1

11) Show the average return time based on the loan limit

This query statement can display the average return time for different loan_type and will return the average return ti me and its loan period. This will help the library to predict when the members are likely to return the resource.

```
SELECT AVG(FLOOR(l.return_date - l.loan_date)) avg_return_time, lt.loan_period
 FROM loan l
  JOIN item_ i ON i.item_id=l.item_id
   JOIN loan_type lt ON lt.loan_type_id=i.loan_type_id
    WHERE l.return_date IS NOT NULL
     GROUP BY lt.loan_period;
```

AVG_RETURN_TIME	LOAN_PERIOD
6.5	14
9	2

12) Show the average overdue time amount paid

This query statement can display the average overdue time of different loan_period and will return the average number of days later and loan_peroid. This can allow librarians to audit the library and consider changing the loan limits of certain resources or buy more resources.

```
SELECT AVG(FLOOR((l.return_date - l.loan_date) - lt.loan_period)) avg_dates_late, lt.loan_period
 FROM loan l
  JOIN item_ i ON i.item_id=l.item_id
   JOIN loan_type lt ON lt.loan_type_id = i.loan_type_id
    WHERE return_date IS NOT NULL
     GROUP BY lt.loan_period;
```

AVG_DATES_LATE	LOAN_PERIOD
-7.5	14
7	2

Creating Triggers

To enforce the rules set out within the coursework specification we have implemented 4 triggers. The triggers are designed to: prohibit loaning a resource due to having an overdue, prohibiting loans due to being suspended, prohibiting a library member borrowing a book when they are at their borrow limit and stopping someone from borrowing a book which is designated for library only.

Trigger 1: prohibit_loan_due_to_res

```
CREATE OR REPLACE TRIGGER prohibit_loan_due_to_overdue_res
BEFORE INSERT ON loan
FOR EACH ROW
DECLARE
has_overdues NUMBER(2);
BEGIN
    SELECT COUNT(member_id) INTO has_overdues
     FROM loan_records
      WHERE return_date IS NULL
      AND SYSDATE >= (loan_date + loan_period)
        AND member_id = :NEW.member_id;
    IF (has_overdues != 0) THEN
     RAISE_APPLICATION_ERROR (-20001, 'Member has been prohibited from borrowing resources due to overdue resources');
    END IF;
END;
/
```

This trigger is designed to raise an error if the library member who is trying to borrow a book currently has an overdue book. Our trigger is specified to fire before the triggering event, in this case before a row is inserted in the loan table, when someone is trying to borrow a book.
We then declare a variable ‘has_overdues’ as a number. For our PL/SQL block we count the number of times said member has an overdue book by checking how many times their member_id is in the loan_records view where the return day is NULL, showing a book borrowed and where the SYSDATE is greater than or equal to the loan_date + loan_period, showing said book is overdue.
Then we check our declared function and if it is not 0 meaning the member has an overdue book we raise the exception with a unique error number and explanatory text.
We have included an example below:
INSERT INTO loan VALUES(20,'LIB005',1100,TO_DATE('15-12-2021', 'DD-MM-YYYY'),NULL);
Leads to the following error message:
ORA-20001: Member has been prohibited from borrowing resources due to overdue resources ORA-06512: at "SQL_ZPKRMJYMVWUKOVUAETJHBFIWF.PROHIBIT_LOAN_DUE_TO_OVERDUE_RES", line 10
ORA-06512: at "SYS.DBMS_SQL", line 1721

We had originally used overdue_member number as the declared variable and attempted to select the member into the declared value, however, we found that if the member did not have any overdue resources, we would encounter a “no data found” error and the exception handling of NO_DATA

_FOUND to be faulty and the trigger would not function unless the exception handling was removed.

Trigger 2: prohibit_loan_due_to_suspension

```
CREATE OR REPLACE TRIGGER prohibit_loan_due_to_suspension
BEFORE INSERT ON loan
FOR EACH ROW
DECLARE
fine_owned NUMBER(2);
BEGIN
SELECT SUM(CASE WHEN l.return_date IS NOT NULL THEN FLOOR(l.return_date - l.loan_date-lp.loan_period)
ELSE FLOOR(SYSDATE - l.loan_date-lp.loan_period) END) INTO fine_owned
FROM fine f
JOIN loan l ON f.loan_id = l.loan_id
JOIN item_ i ON l.item_id = i.item_id
JOIN loan_type lp ON i.loan_type_id = lp.loan_type_id
WHERE f.paid_date IS NULL AND l.member_id = :NEW.member_id;
IF (fine_owned  >= 10) THEN
RAISE_APPLICATION_ERROR (-20002, 'Member has been prohibited from borrowing resources due to fine suspension');
    END IF;
END;
/
```

The second trigger is designed to stop a library member from borrowing a book if they are suspended from having fines or $10 or over. Like the previous trigger it is designed to trigger before a row is inserted on the loan table, therefore before someone borrows a book. We declare the variable fine_owned and then in our PL/SQL code block we check for fines which have not been paid that match the member_id of our library member, we then calculate the fines using the return date if the resource has been returned of the sysdate if the resource has not been returned, and add up all the fines that the library member has into our declared fine_owned. If the member's fines are at or above $10 they are then the system raises an error.

We have included an example below:

INSERT INTO loan VALUES(21,'LIB004',1101,TO_DATE('15-12-2021', 'DD-MM-YYYY'),NULL);

Leads to the following error message:

ORA-20002: Member has been prohibited from borrowing resources due to fine suspension ORA-06512: at "SQL_OVUVLIPAKMCMXAMRBKHBXLOVP.PROHIBIT_LOAN_DUE_TO_SUSPENSION", line 11
ORA-06512: at "SYS.DBMS_SQL", line 1721

Trigger 3: prohibit_loan_due_to_borrow_limit

```
CREATE OR REPLACE TRIGGER prohibit_loan_due_to_borrow_limit
BEFORE INSERT ON loan
FOR EACH ROW
DECLARE
```

```
current_total_loan NUMBER(2);
loan_limit NUMBER(2);
BEGIN
SELECT COUNT(member_id) INTO current_total_loan FROM loan_records WHERE return_date IS NULL AND member_id = :NEW.member_id;
SELECT loan_limit INTO loan_limit FROM member_records WHERE member_id = :NEW.member_id;
IF (current_total_loan >= loan_limit) THEN
RAISE_APPLICATION_ERROR (-20003, 'Member has been prohibited from borrowing resources due to exceeding maximum loan limit');
END IF;
END;
/
```

Our third trigger is designed to stop library members from borrowing books that are over their borrow limit, which is 5 for students and 10 for staff. As with our previous two triggers it is designed to trigger before a row is inserted on the loan table, so before someone borrows a book. In this trigger we define 2 variables of current_total_loan and loan_limit. In our PL/SQL block we count the instances of resources which the library member has currently borrowed and put it into our current_total_loan variable, then we take the loan_limit for the member and put it into loan_limit. We then compare the 2 values and if the library member is at or above (to account for errors) their loan_limit then an application error is raised.

We have included an example below (4 books are borrowed to get the student to their limit):

```
INSERT INTO loan VALUES(22,'LIB002',1102,TO_DATE('15-12-2021', 'DD-MM-YYYY'),NULL);
INSERT INTO loan VALUES(23,'LIB002',1106,TO_DATE('15-12-2021', 'DD-MM-YYYY'),NULL);
INSERT INTO loan VALUES(24,'LIB002',1107,TO_DATE('15-12-2021', 'DD-MM-YYYY'),NULL);
INSERT INTO loan VALUES(25,'LIB002',1108,TO_DATE('15-12-2021', 'DD-MM-YYYY'),NULL);
INSERT INTO loan VALUES(26,'LIB002',1109,TO_DATE('15-12-2021', 'DD-MM-YYYY'),NULL);
```

Leads to the following error message:

```
1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.
1 row(s) inserted.
ORA-20003: Member has been prohibited from borrowing resources due to exceeding maximum loan limit ORA-06512: at "SQL_OVUVLIPAKMCMXAMRBKHBXLOVP.PROHIBIT_LOAN_DUE_TO_BORROW_LIMIT", line 8
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

Trigger 4: resource_not_available_to_loan

```
CREATE OR REPLACE TRIGGER resource_not_available_to_loan
BEFORE INSERT ON loan
FOR EACH ROW
DECLARE
limit_ NUMBER(2);
BEGIN
SELECT lt.loan_period INTO limit_
FROM loan_type lt
JOIN item_ i ON lt.loan_type_id = i.loan_type_id
where :NEW.item_id=item_id;
```

```
IF (limit_ = 0) THEN
RAISE_APPLICATION_ERROR (-20004, 'This resource cannot be borrowed');
END IF;
END;
/
```

Our last trigger is designed to stop library members from borrowing resources which are not available for borrowing. As with our previous triggers it is designed to trigger before a row is inserted on the loan table, so before someone borrows a resource. In this trigger we define the variable limit_. Then in our PL/SQL block we check the loan_limit which matches our item and import it into our declared function. We can then check and if the limit_ field is zero then an application error is raised.

We have included an example below

```
INSERT INTO loan VALUES(27,'LIB003',1109,TO_DATE('15-12-2021', 'DD-MM-YYYY'),NULL);
```

Leads to the following error message:

```
ORA-20004: This resource cannot be borrowed ORA-06512: at "SQL_OVUVLIPAKMCMXAMRBKHBXLOVP.RESOURCE_NOT_AVAILABLE_TO_LOAN", line 9
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

Database Security Considerations:

As with any parts of an organisation we have to be careful with the database to not allow bad actors access from a strategic, legal and ethical position. A strategic sense in that corporate data has information about their activities and other organisations could gain a competitive advantage with access to the data. In the legal sense, organisations hold private data of individuals and therefore are beholden to the UK’s Data Protection of 2018. From an ethical sense it is important for the organisation to “do what is right” regarding the data based on their/societies values and expectations, this comes to a head more in cases where there are legal grey areas, where the law has not yet caught up. It can come into conflict as for organisations it is often important to spend enough money not to be conducting illegal business but the incentives dry up in a strategic sense, for instance “this security is good enough, why spend more to protect users’ data if it will not make us money?”. In these cases the ethical considerations can step in.

Of course for our college library database the incentives are different. If we consider it to be a UK university, they will be non-profit institutions (apart from BPP University and the University of Law [1]), and will therefore have less strategic incentives to be as cut-throat as other corporations. However, they will obviously hold a wealth of private data which could be used for nefarious purposes.

The end goal of our database security is to ensure confidentiality, integrity and availability. Confidentiality ensures that people who should not have access to the data are unable to view it. Integrity ensures that people who should not be able to modify data are unable to edit it. Availability ensures that people who should have access are able to view and edit as appropriate.

In the next few paragraphs we will be looking at different security implementations to maintain the integrity of our database and discussing the various merits of implementing them for our system. When discussing security of our database it is important to first think about the routes of access that threats have for accessing our data.

The first category of threats are hardware threats, these can include theft of hardware containing data and damage to equipment. Damage to equipment can be due to natural causes such as floods or storms, inadequate protection such as due to fire or loss of power, or maliciousness through sabotage. There are a lot of countermeasures which are unfortunately beyond the scope of protections that we can implement but we can implement encryption to protect data after theft, and we can utilise backups and RAID technology to ameliorate damage to hardware.

Another category of threat is Database threats. These can be unauthorised access to data leading to theft or corruption of data. We can counter these threats by enforcing authorisation requirements prior to access to the database, access controls and view implementation so that employees are only able to access data which is relevant to their work.

The next category of threats are attacked from the DBMS and Application software. In this case the DBMS or application security can be bypassed, leading to alteration or theft of data. Count ermeasures to these are unfortunately beyond the scope of what we are able to do beyond keeping our DBMS and Application up to date as much as possible.

The next threat category is users, where there is some overlap with the database vector. Users may be able to view and copy unauthorised data, they could have errant editing rights meaning that they could delete data. There are also social engineering methods of gaining access to a user’ s account. We can again counter these threats with authorisation, access controls, views. Methods such as exploiting users to gain login credentials are beyond the scope of what we are able to implement, but user training can be helpful.

We will now discuss the methods that we can implement and are within our scope. The first of which is the act of encryption. This is where our data is encoded with a “public key” and the database is unreadable except by users who have the “private key” to decrypt the data. There is a performance overhead associated with it as the host computer has to decrypt the database on access and then encrypt any changes. The more complex the encryption the larger the overhead. Varying amounts of the database can be encrypted from select columns to the whole database, larger overhead the more of the data is encrypted. However, we believe that the performance downgrade is worth it to ensure that our data is more secure. We would do this by implementing Oracle’ s Transparent Data Encryption feature [2].

The next feature we suggested was backups, where we can store a copy of our database. It is ideal to have our database as offline so it is not able to be accessed by anyone wanting to compromise the data and having it off site so that in the event of an accident such as a flood there will only be 1 copy of the data which is lost (either the database location or backup location should be safe). Again we feel this is an important measure to implement although we may not be able to ensure it is both offline and offsite. We are able to implement this through Oracle’ s Recovery Manager Utility [3].

RAID is an acronym that stands for Redundant Array of Independent Disks. Described by Patterson et al. in their 1988 paper they described a system which would separate data across several hard disks [4]. And allow for some of those hard disks to fail without any loss of data. We would be supportive of this but it is out of our scope to implement.

Another countermeasure is to require authorisation prior to access to data. This is commonly accomplished with a login screen requiring a username and password to access the database. We would also advocate for the use of authorisation for access to the database, which we can do in Oracle through their local authorisation system or by piggybacking onto an OS authentication, which the college likely already has [5].

The next feature that we discussed earlier is in access controls. This is where we are able to grant or revoke users the ability to access database objects, and whether they are able to view or edit those objects. We should try to provide each user the minimal amount of privileges to perform their job role. The objects that we are able to provide access to include relations and views. Through oracle we are able to limit database object control down the column and with actions we are able to have different privileges for SELECT, INSERT, UPDATE and DELETE allowing finer control, using schema object or system privileges. For example we are able to use the command:

GRANT X ON Y TO Z;

Where x is the command, Y is the database object and z is the user we want to grant access to, and we can use the REVOKE command in the same way to remove these privileges [6].

Finally we have the views feature, this is where we create a table of data, which is carefully selected based on other tables, this allows for increased security as we are able to grant a user access to only the view but not the underlying data, meaning that we are able to hide more sensitive data. In our views we have tried to also adhere to these principles.

Bibliography

[1] C. Cook, "UK approves first for-profit university," in *Financial Times*, ed. London: The Financial Times Limited, 2012.

[2] P. Huey, "Introduction to Transparent Data Encryption," in *Oracle® Database Advanced Security Guide*, 12.1 ed.: Oracle, 2017, pp. 21-27.

[3] M. Cyran, "Introduction to Backup," in *Oracle® Database Concepts*, vol. 10.2: Oracle, 2005, ch. 15, pp. 151-156.

[4] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, 1988, pp. 109-116.

[5] Oracle. "Database Authentication and Authorization." https://www.oracle.com/database/technologies/security/db-authn-authz.html (accessed 2021).

[6] S. Jeloka, "Authorization: Privileg es, Roles, Profiles, and Resource Limitations," in *Oracle® Database Security Guide*, 10.2 ed. (Oracle® Database: Oracle, 2012, ch. 5, pp. 5-1 to 5-24.