# ECS740P - DATABASE SYSTEMS

## Coursework 2

## College Library System

Group 3

Introduction 2

Relational Schema 2

Creating Tables 4

Sample Test Data 8

Creating Views 13

Queries 19

Creating Triggers 26

Data Security Issues 34

References 35

**Introduction**

This report aims to implement the relational schema of the College Library System designed in Coursework 1. The database application will be implemented by detailing the following:

1. A recap on the relational schema
2. SQL commands to create the tables of the database
3. Sample test data for the database
4. Views for the database
5. SQL queries and explanations
6. Triggers for the database
7. Data security issues of the database

The DBMS chosen is Oracle Live SQL, and the scripts presented in a separate file of the report will be aligned with the DBMS.

Relational schema

In Coursework 1, we have concluded with the following data base schema:

**User** (libcard\_id, last\_name, first\_name, country\_code, mobile\_number, user\_type, user\_status)

**User\_Limit** (user\_type, borrowing\_limit)

**Resource** (resource\_id, resource\_type, resource\_class, title)

**Resource\_shelf** (resource\_class, shelf\_number)

**Location** (shelf\_number, floor\_number)

**Copy** (resource\_id, copy\_number, loan\_period)

**Print\_info** (resource id, publisher, isbn/issn)

**Print\_author** (resource id, author)

**Multimedia\_Producer** (resource id, producer)

**Multimedia\_Length** (resource id, length)

**Loan** (library card id, resource id, copy number, date\_borrow, date\_return)

**Payment** (libcard\_id, resource\_id, copy\_number, date\_borrow, date\_return, payment\_amount)

Before proceeding to build the tables, we needed to make certain changes with the schema.

Firstly, we are changing some of the table names to escape Oracle’s reserved words, hence User is becoming Users and Resource is becoming Resources. Moreover, some of the table names are shortened to make queries easier and cleaner.

Secondly, Loan and Payment tables have four-attribute primary keys. As the natural primary keys are cumbersome, we are introducing surrogate keys in the two tables to act as primary keys, such that both tables will no longer have composite primary keys.

After making the modifications, the final schema is as follow:

**User** (libcard\_id, last\_name, first\_name, country\_code, mobile\_number, user\_type, user\_status)

**User\_limit** (*user\_type*, borrowing\_limit)

**Resources** (resource\_id, resource\_type, resource\_class, title)

**Resource\_shelf** (*resource\_class*, *shelf\_number*)

**Location** (shelf\_number, floor\_number)

**Copy** (*resource\_id*, copy\_number, loan\_period)

**Print\_info** (*resource id*, publisher, isbn/issn)

**Print\_author** (*resource id*, author)

**MM\_info** (*resource id*, producer)

**MM\_length** (*resource id*, length)

**Loan** (loan\_id, *library card id, resource id, copy number, date\_borrow*, date\_return)

**Payment** (*loan\_id*, payment\_amount)

**Creating Tables**

Due to foreign key constraints, we have re-ordered the sequence of table creation such that tables being referenced are created first.

**User\_limit**

The table is created first as there is no depending column. The attributes are user\_type, which should be a string, and borrowing\_limit, a number. The primary key of the table would be the user\_type.

CREATE TABLE USER\_LIMIT(

USER\_TYPE VARCHAR2(10),

BORROWING\_LIMIT NUMBER(2) NOT NULL,

CONSTRAINT PK\_USERLIMIT PRIMARY KEY (USER\_TYPE)

);

**Users**

The USERS table contains seven columns, where all of them belong to different lengths of strings. Although several of the columns have their values as strings of numbers, as we do not have to perform calculations on them, it is best to set their datatypes to VARCHAR2. The primary key of the table is set to the libcard\_id, and the only foreign key in this table is the user\_type, referencing the above User\_limit table.

There are a couple of columns, last\_name, first\_name and user\_status, since their values are critical, they are set to have NOT NULL constraints. The CHECK constraint on the USER\_STATUS is used for limiting the input to one of ‘Active’, ‘Deactivated’, or ‘Suspended’.

CREATE TABLE USERS(

LIBCARD\_ID VARCHAR2(7),

LAST\_NAME VARCHAR2(10) NOT NULL,

FIRST\_NAME VARCHAR2(10) NOT NULL,

COUNTRY\_CODE VARCHAR2(3),

MOBILE\_NUMBER VARCHAR2(11),

USER\_TYPE VARCHAR2(10),

USER\_STATUS VARCHAR2(12) NOT NULL,

CONSTRAINT PK\_USERS PRIMARY KEY (LIBCARD\_ID),

CONSTRAINT FK\_USERS FOREIGN KEY (USER\_TYPE) REFERENCES USER\_LIMIT(USER\_TYPE),

CONSTRAINT CHECK\_STATUS CHECK (USER\_STATUS='Active' OR USER\_STATUS='Deactivated' OR USER\_STATUS='Suspended')

);

**Location**

The LOCATION table has two attributes, shelf\_number and floor\_number, with strings of numbers for their value. Again, since we do not have to perform calculations on the numbers, we set the datatypes to VARCHAR2. Since each shelf must be present on a designated floor number, floor\_number is constraint to NOT NULL, and shelf\_number is set to be the primary key.

CREATE TABLE LOCATION(

SHELF\_NUMBER VARCHAR2(4),

FLOOR\_NUMBER VARCHAR2(1) NOT NULL,

CONSTRAINT PK\_LOCATION PRIMARY KEY (SHELF\_NUMBER)

);

**Resource\_Shelf**

The Resource\_shelf table has both attributes resource\_class and shelf\_number as the primary keys, as one resource\_class, i.e., a large genre can span across multiple shelves, and multiple small genres could be on the same shelf. The shelf\_number is a foreign key with reference to the above location table.

CREATE TABLE RESOURCE\_SHELF(

RESOURCE\_CLASS VARCHAR2(30),

SHELF\_NUMBER VARCHAR2(4),

CONSTRAINT PK\_RESOURCESHELF PRIMARY KEY(RESOURCE\_CLASS, SHELF\_NUMBER),

CONSTRAINT FK\_RESOURESHELF FOREIGN KEY(SHELF\_NUMBER) REFERENCES LOCATION(SHELF\_NUMBER)

);

**Resources**

The Resource table includes four attributes, the primary key resource\_id, the unique identifier of each resource, the resource\_type and resource\_class mentioned above, and the title of the resource. The title of the resource, being a key information, is set to NOT NULL to ensure each resource contains such value.

CREATE TABLE RESOURCES(

RESOURCE\_ID VARCHAR2(8),

RESOURCE\_TYPE VARCHAR2(10),

RESOURCE\_CLASS VARCHAR2(30),

TITLE VARCHAR2(100) NOT NULL,

CONSTRAINT PK\_RESOURCES PRIMARY KEY (RESOURCE\_ID)

);

**Copy**

The Copy table contains three attributes, the composite primary keys of resource\_id and copy\_number, where resource\_id is also a foreign key referencing the above Resources table, and the attribute of loan\_period. There is a CHECK constraint for the loan\_period column, where the value should be between -1 and 14. These are the days of the maximum loan period for each copy. -1 indicates the copy is not allowed to be borrowed, while 14 indicates the copy could be borrowed for a maximum of 14 days before it is due.

CREATE TABLE COPY (

RESOURCE\_ID VARCHAR2(8) NOT NULL,

COPY\_NUMBER VARCHAR2(2) NOT NULL,

LOAN\_PERIOD NUMBER(2) NOT NULL,

CONSTRAINT PK\_COPY PRIMARY KEY (RESOURCE\_ID, COPY\_NUMBER),

CONSTRAINT FK\_COPY FOREIGN KEY(RESOURCE\_ID) REFERENCES RESOURCES (RESOURCE\_ID),

CONSTRAINT COPY\_LOANPERIOD CHECK (LOAN\_PERIOD BETWEEN -1 AND 14)

);

**Print\_info**

This table includes Resource\_ID, the primary key as well as a foreign key, and two other attributes exclusive to the subclass of “print resources”, namely the publisher and the isbn/issn of the print resource. When a resource is being deleted in the Resource table, the information of that resource in this table is no longer needed to be store, hence the ON DELETE CASCADE option is added to the foreign key constraint.

CREATE TABLE PRINT\_INFO(

RESOURCE\_ID VARCHAR2(8),

PUBLISHER VARCHAR2(30),

ISBN\_ISSN VARCHAR2(13) UNIQUE,

CONSTRAINT PK\_PRINTINFO PRIMARY KEY (RESOURCE\_ID),

CONSTRAINT FK\_PRINTINFO FOREIGN KEY (RESOURCE\_ID) REFERENCES RESOURCES (RESOURCE\_ID) ON DELETE CASCADE

);

**Print\_author**

This table illustrates the relationship between different print resources and their authors. Since each resource can be co-authored by multiple authors and one author can have multiple works, both resource\_id and author are set to be the primary keys. The resource\_id is of course a foreign key referencing the Resources table.

CREATE TABLE PRINT\_AUTHOR(

RESOURCE\_ID VARCHAR2(8),

AUTHOR VARCHAR2(20),

CONSTRAINT PK\_PRINTAUTHOR PRIMARY KEY (RESOURCE\_ID, AUTHOR),

CONSTRAINT FK\_PRINTAUTHOR FOREIGN KEY (RESOURCE\_ID) REFERENCES RESOURCES (RESOURCE\_ID) ON DELETE CASCADE

);

**MM\_info**

Similar to Print\_author, MM\_info has a composite primary key of resource\_id and producer for the same reasons.

CREATE TABLE MM\_INFO(

RESOURCE\_ID VARCHAR2(8),

PRODUCER VARCHAR2(30),

CONSTRAINT PK\_MMINFO PRIMARY KEY (RESOURCE\_ID, PRODUCER),

CONSTRAINT FK\_MMINFO FOREIGN KEY (RESOURCE\_ID) REFERENCES RESOURCES (RESOURCE\_ID) ON DELETE CASCADE

);

**MM\_length**

The MM\_length table, on the other hand, indicates the length of each multimedia resource. The Length column has the datatype of number with maximum 3 digits, which is the length of the multimedia resource in minutes. The primary is again the resource\_id, which is also a foreign key referencing the Resources table.

CREATE TABLE MM\_LENGTH(

RESOURCE\_ID VARCHAR2(8),

LENGTH NUMBER(3) NOT NULL,

CONSTRAINT PK\_MMLENGTH PRIMARY KEY (RESOURCE\_ID),

CONSTRAINT FK\_MMLENGTH FOREIGN KEY (RESOURCE\_ID) REFERENCES RESOURCES (RESOURCE\_ID) ON DELETE CASCADE

);

**Loan**

We have introduced a surrogate primary key, the loan\_id, for the Loan table, replacing the role of the natural candidate keys of libcard\_id, resource\_id, copy\_number, and date\_borrow combined. The loan\_id will generate on its own as a new number every time a new tuple is added to the table. The libcard\_id is a foreign key referencing the Users table while the resource\_id and copy\_number references the Copy table. The date\_borrow is set to NOT NULL, as it is a key information of the loan record. Finally, when the resource\_id or copy\_number is set to ON DELETE SET NULL, such that when a resource or a copy is removed, the loan record could still be seen as a user borrowing a copy that no longer exists, instead of wiping out the whole record.

CREATE TABLE LOAN(

LOAN\_ID NUMBER GENERATED AS IDENTITY,

LIBCARD\_ID VARCHAR2(7),

RESOURCE\_ID VARCHAR2(8),

COPY\_NUMBER VARCHAR2(2),

DATE\_BORROW DATE NOT NULL,

DATE\_RETURN DATE,

CONSTRAINT PK\_LOAN PRIMARY KEY (LOAN\_ID),

CONSTRAINT FK\_LOAN\_LIBCARDID FOREIGN KEY (LIBCARD\_ID) REFERENCES USERS(LIBCARD\_ID),

CONSTRAINT FK\_LOAN\_COPY FOREIGN KEY (RESOURCE\_ID, COPY\_NUMBER) REFERENCES COPY(RESOURCE\_ID, COPY\_NUMBER) ON DELETE SET NULL

);

**Payment**

Finally, with the purpose of auditing all fine payments, the payment table will have a primary key of loan\_id (referencing a loan record in the Loan table) and the payment\_amount(the amount of fine payment received for that loan record).

CREATE TABLE PAYMENT(

LOAN\_ID NUMBER,

PAYMENT\_AMOUNT NUMBER (4) NOT NULL,

CONSTRAINT PK\_PAYMENT PRIMARY KEY (LOAN\_ID),

CONSTRAINT FK\_PAYMENT FOREIGN KEY (LOAN\_ID) REFERENCES LOAN(LOAN\_ID)

);

**Sample Test Data**

This section lists the sample test data used for each table, with the data inserting scripts detailed in the separate PLSQL script file.

**User Limit**

|  |  |
| --- | --- |
| **USER\_TYPE** | **BORROWING\_LIMIT** |
| Student | 5 |
| Staff | 10 |

**Users**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **LIBCARD\_ID** | **LAST\_NAME** | **FIRST\_NAME** | **COUNTRY\_CODE** | **MOBILE\_NUMBER** | **USER\_TYPE** | **USER\_STATUS** |
| 9538839 | Kemmis | Loadin | 44 | 12900789282 | Student | Active |
| 8592723 | Theodora | Kantor | 44 | 27777068036 | Student | Active |
| 3239001 | Bealle | Diperaus | 44 | 50533368146 | Student | Deactivated |
| 4282801 | Latisha | Pearson | 44 | 49501947110 | Student | Suspended |
| 5739753 | Michaella | Strafen | 353 | 14370969 | Student | Active |
| 2830482 | Thibaut | Thorold | 44 | 62051043520 | Student | Active |
| 8083280 | Thorsten | Glenn | 44 | 58788472692 | Student | Active |
| 2382819 | Jemmy | Wash | 44 | 3262828952 | Staff | Active |
| 7171742 | Bary | Medley | 44 | 76023729185 | Staff | Active |
| 6177322 | Barbie | Lukasik | 358 | 508381212 | Staff | Active |

**Location**

|  |  |
| --- | --- |
| **SHELF\_NUMBER** | **FLOOR\_NUMBER** |
| 1004 | 1 |
| 1011 | 1 |
| 1024 | 1 |
| 1027 | 1 |
| 1032 | 1 |
| 1049 | 1 |
| 1052 | 1 |
| 1054 | 1 |
| 1063 | 1 |
| 1070 | 1 |
| 1080 | 1 |
| 1082 | 1 |
| 1091 | 1 |
| 1094 | 1 |
| 2017 | 2 |
| 2019 | 2 |
| 2026 | 2 |
| 2040 | 2 |
| 2043 | 2 |
| 2045 | 2 |
| 2056 | 2 |
| 2057 | 2 |
| 2092 | 2 |
| 3051 | 3 |
| 3073 | 3 |
| 3074 | 3 |
| 3077 | 3 |
| 3084 | 3 |
| 3095 | 3 |

**Resource\_shelf**

|  |  |
| --- | --- |
| **RESOURCE\_CLASS** | **SHELF\_NUMBER** |
| Art/architecture | 2017 |
| Autobiography | 2019 |
| Biography | 2026 |
| Business and economics | 1094 |
| Business/economics | 2040 |
| Classic Fiction | 1004 |
| Classic Fiction | 1011 |
| Classic Fiction | 1024 |
| Cookbook | 2045 |
| Crafts/hobbies | 2043 |
| Crime | 1049 |
| Encyclopedia | 1091 |
| Health | 1082 |
| Home and garden | 1082 |
| Journal | 1070 |
| Journal | 1080 |
| Math | 2057 |
| Math | 2092 |
| Philosophy | 2056 |
| Poetry | 1049 |
| Religion | 1082 |
| Science | 3051 |
| Science | 3073 |
| Science | 3074 |
| Science | 3077 |
| Science | 3084 |
| Science | 3095 |
| Science Fiction | 1027 |
| Science Fiction | 1032 |
| Short Story | 1049 |
| Sports and leisure | 1082 |
| Textbook | 1052 |
| Textbook | 1054 |
| Textbook | 1063 |

**Resources**

|  |  |  |  |
| --- | --- | --- | --- |
| **RESOURCE\_ID** | **RESOURCE\_TYPE** | **RESOURCE\_CLASS** | **TITLE** |
| 27369096 | Book | Business/economics | The Lean Startup |
| 23315921 | Book | Dictionary | Macmillan English Dictionary for Advanced Learners |
| 88147914 | Book | Journal | The Truth about Cancer |
| 17649202 | Book | Math | The Principia: Mathematical Principles of Natural Philosophy |
| 35537314 | Book | Science | Introduction to Quantum Mechanics |
| 82120851 | Book | Science | Fundamentals of Aerodynamics |
| 18736414 | Book | Biography | Jim Henson: The Biography |
| 85815155 | Book | Textbook | UNIX and Linux System Administration Handbook |
| 43106382 | Book | Art/architecture | Between the Lines |
| 63290045 | Book | Encyclopedia | The Encyclopedia of Gods |
| 13951381 | Book | Classic Fiction | Huge Dorrit |
| 20174519 | CD | History | Helen of Troy |
| 48527649 | CD | Religion | The History of God |
| 47452487 | CD | Science Fiction | Fated Blades |
| 87386855 | DVD | Fantasy | Jade Legacy |
| 19270142 | DVD | Sports and leisure | Best Goals Scored |
| 66138583 | DVD | Home and garden | Japanese Style Gardens |
| 31512743 | Video | Crime | Tinker Tailor Soldier Spy |
| 09496244 | Video | Thriller | Crossfire |
| 49873226 | Video | Health | Tackling Back Pain |

**Copy**

|  |  |  |
| --- | --- | --- |
| **RESOURCE\_ID** | **COPY\_NUMBER** | **LOAN\_PERIOD** |
| 27369096 | 1 | 14 |
| 31512743 | 1 | 14 |
| 31512743 | 2 | 14 |
| 87386855 | 1 | 14 |
| 87386855 | 2 | 2 |
| 23315921 | 1 | 14 |
| 09496244 | 1 | 14 |
| 19270142 | 1 | 14 |
| 48527649 | 1 | 14 |
| 88147914 | 1 | 14 |
| 49873226 | 1 | 14 |
| 47452487 | 1 | 14 |
| 17649202 | 1 | 14 |
| 35537314 | 1 | 2 |
| 82120851 | 1 | 14 |
| 18736414 | 1 | 14 |
| 18736414 | 2 | 14 |
| 85815155 | 1 | 14 |
| 85815155 | 2 | 14 |
| 85815155 | 3 | 14 |
| 85815155 | 4 | 14 |
| 85815155 | 5 | 14 |
| 85815155 | 6 | 14 |
| 85815155 | 7 | 14 |
| 85815155 | 8 | 2 |
| 85815155 | 9 | 2 |
| 85815155 | 10 | 2 |
| 85815155 | 11 | -1 |
| 43106382 | 1 | 2 |
| 63290045 | 1 | -1 |
| 13951381 | 1 | 14 |
| 13951381 | 2 | 14 |
| 13951381 | 3 | 14 |
| 13951381 | 4 | 14 |

**Print\_info**

|  |  |  |
| --- | --- | --- |
| **RESOURCE\_ID** | **PUBLISHER** | **ISBN\_ISSN** |
| 27369096 | Rlg Hunting Stands | 9781234567897 |
| 23315921 | Gross Publications | 9310831842018 |
| 88147914 | Dominique Olbrechts | 5830813792471 |
| 17649202 | Edi Multimedia | 9541847279186 |
| 35537314 | Chiworks | 3241624917491 |
| 82120851 | Smith Printing CO | 1294719419971 |
| 18736414 | Sidney Herald | 2312083128058 |
| 85815155 | The Atlanta Tribune | 3180538714538 |
| 43106382 | Mindlab Media Inc | 5398164287101 |
| 63290045 | Alaska Register | 3197752917420 |
| 13951381 | Alaska Register | 5748164967101 |

**Print\_author**

|  |  |
| --- | --- |
| **RESOURCE\_ID** | **AUTHOR** |
| 13951381 | T. Adams |
| 17649202 | L. Tolstoy |
| 18736414 | L. Tolstoy |
| 18736414 | M. Hill |
| 23315921 | A. Hopkins |
| 27369096 | A. Hopkins |
| 27369096 | J. Wong |
| 35537314 | L. Tolstoy |
| 35537314 | R. Howard |
| 43106382 | M. Abe |
| 63290045 | J. Ackerman |
| 82120851 | L. Tolstoy |
| 82120851 | R. Tagore |
| 85815155 | D. Bains |
| 85815155 | L. Tolstoy |
| 85815155 | L. Urban |
| 85815155 | Y. Agnew |
| 88147914 | E. Dickinson |

**MM\_info**

|  |  |
| --- | --- |
| **RESOURCE\_ID** | **PRODUCER** |
| 09496244 | Universal Studios |
| 19270142 | Pan Macmillan |
| 20174519 | Comcast |
| 31512743 | Walt Disney |
| 47452487 | Hachette Livre |
| 48527649 | Sony |
| 49873226 | Fox |
| 66138583 | Bloomsbury |
| 87386855 | Pixar |

**MM\_length**

|  |  |
| --- | --- |
| **RESOURCE\_ID** | **LENGTH** |
| 20174519 | 45 |
| 48527649 | 47 |
| 47452487 | 52 |
| 87386855 | 95 |
| 19270142 | 126 |
| 66138583 | 155 |
| 31512743 | 30 |
| 49873226 | 28 |
| 09496244 | 43 |

**Loan**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LOAN\_ID** | **LIBCARD\_ID** | **RESOURCE\_ID** | **COPY\_NUMBER** | **DATE\_BORROW** | **DATE\_RETURN** |
| 1 | 5739753 | 35537314 | 1 | 30-AUG-21 | 01-SEP-21 |
| 2 | 2382819 | 17649202 | 1 | 31-AUG-21 | 02-SEP-21 |
| 3 | 3239001 | 82120851 | 1 | 01-SEP-21 | 15-SEP-21 |
| 4 | 3239001 | 18736414 | 2 | 01-SEP-21 | 15-SEP-21 |
| 5 | 2830482 | 43106382 | 1 | 02-SEP-21 | 04-SEP-21 |
| 6 | 6177322 | 85815155 | 6 | 04-SEP-21 | 20-SEP-21 |
| 7 | 9538839 | 85815155 | 3 | 05-SEP-21 | 07-SEP-21 |
| 8 | 9538839 | 13951381 | 2 | 06-OCT-21 | 20-OCT-21 |
| 9 | 8592723 | 13951381 | 4 | 08-OCT-21 | 22-OCT-21 |
| 10 | 8083280 | 87386855 | 1 | 15-NOV-21 | 08-DEC-21 |
| 11 | 5739753 | 85815155 | 3 | 22-NOV-21 | - |
| 12 | 8592723 | 17649202 | 1 | 04-DEC-21 | - |

**Payment**

|  |  |
| --- | --- |
| **LOAN\_ID** | **PAYMENT\_AMOUNT** |
| 6 | 2 |

Apart from inserting the above normal data, further tests are conducted to make sure the written column constraints are working properly and the expected errors are being returned by the DBMS. Here are some examples:

INSERT INTO Users VALUES ('6767321','Southgate','Max','44','9595531234','Stdent','Inactive'); -- Violating CHECK constraint on the status

INSERT INTO Copy VALUES ('27369096','2',19); -- Violating CHECK constraint on the loan period

INSERT INTO Resources VALUES ('10008113','Book','Classic Fiction', NULL); -- Violating NOT NULL constraint on title

Foreign key constraints have also been tested, for example we try to delete a resource where a child copy is present, violating the foreign key integrity:

DELETE FROM RESOURCES WHERE RESOURCE\_ID = '88147914' -- Error is returned

Also testing the ON DELETE CASCADE constraints in tables Print\_info, Print\_author, MM\_info and MM\_length, this time by deleting the children copies of the resources first, then deleting the resource, and query the related tables. A print resource #88147914 and a multimedia resource #20174519 are put to test:

DELETE FROM Copy WHERE resource\_id = '88147914' OR resource\_id = '20174519';

DELETE FROM RESOURCES WHERE RESOURCE\_ID = '88147914' OR RESOURCE\_ID = '20174519';

Subsequently, the below queries all return “no data found”.

SELECT \* FROM RESOURCES WHERE RESOURCE\_ID = '88147914' OR RESOURCE\_ID = '20174519';

SELECT \* FROM PRINT\_INFO WHERE RESOURCE\_ID = '88147914';

SELECT \* FROM PRINT\_AUTHOR WHERE RESOURCE\_ID = '88147914';

SELECT \* FROM MM\_INFO WHERE RESOURCE\_ID = '20174519';

SELECT \* FROM MM\_LENGTH WHERE RESOURCE\_ID = '20174519';

The ON DELETE SET NULL constraint in the Loan table is also tested, by deleting a particular copy which appeared in a historic loan record with loan\_id #4.

DELETE FROM Copy WHERE resource\_id = '18736414' AND copy\_number = '2';

The query SELECT \* FROM Loan WHERE Loan\_ID = '4'; then returns the following output:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LOAN\_ID** | **LIBCARD\_ID** | **RESOURCE\_ID** | **COPY\_NUMBER** | **DATE\_BORROW** | **DATE\_RETURN** |
| 4 | 3239001 | - | - | 01-SEP-21 | 15-SEP-21 |

**Creating Views**

Three views, all\_copies, all\_loans, and all\_users are created to facilitate the users’ utilisation of the database system.

**All\_copies**

The view all\_copies utilised case statements, subqueries and a number of joins, including inner join, left outer join and full outer join to create a conglomerate view of all copies of all resources, all their details, and their respective statuses.

With this view, all users, including students, staffs and librarians would be able to search for copies of resources, all the particulars including the resource ID, copy number, title, resource type, genre, maximum loan period, location. Subclass specific attributes are also included, such as publisher, ISBN/ISSN and author for print resources, and producer and length for multimedia resources.

Status of each copy, including whether the copy is loaned out, and if yes, whether they are overdue, and the due date of the copy are also listed for the users’ easy reference.

This view could be useful as a single source of information for all resources and copies, since after normalisation, details of each resource and each copy are being stored in different small tables. Below are the SQL scripts for creating the all\_copies view.

CREATE VIEW all\_copies AS

WITH LO AS (SELECT LO.LOAN\_ID, LO.RESOURCE\_ID, LO.COPY\_NUMBER, CO.LOAN\_PERIOD,

CASE WHEN LO.date\_return IS NULL THEN 'YES' END AS Loaned,

CASE WHEN LO.date\_return IS NULL AND LO.DATE\_BORROW + CO.LOAN\_PERIOD < SYSDATE THEN 'YES' END AS Overdue,

CASE WHEN LO.DATE\_RETURN IS NULL THEN LO.DATE\_BORROW + CO.LOAN\_PERIOD END AS DUE\_DATE

FROM LOAN LO

INNER JOIN COPY CO

ON LO.RESOURCE\_ID = CO.RESOURCE\_ID AND LO.COPY\_NUMBER = CO.COPY\_NUMBER

WHERE LO.date\_return IS NULL

ORDER BY LOAN\_ID)

SELECT RE\_CO\_SH.RESOURCE\_ID, RE\_CO\_SH.TITLE, RE\_CO\_SH.RESOURCE\_TYPE, RE\_CO\_SH.RESOURCE\_CLASS, RE\_CO\_SH.COPY\_NUMBER, RE\_CO\_SH.LOAN\_PERIOD, RE\_CO\_SH.SHELVES, RE\_CO\_SH.PUBLISHER, RE\_CO\_SH.ISBN\_ISSN, RE\_CO\_SH.AUTHORS, RE\_CO\_SH.PRODUCER, RE\_CO\_SH.LENGTH, LO.LOANED, LO.OVERDUE, LO.DUE\_DATE

FROM (

SELECT C.RESOURCE\_ID, C.TITLE, C.RESOURCE\_TYPE, C.RESOURCE\_CLASS, C.COPY\_NUMBER, C.LOAN\_PERIOD, SH.SHELVES, C.PUBLISHER, C.ISBN\_ISSN, AUT.AUTHORS, C.PRODUCER, C.LENGTH

FROM (

SELECT RE.RESOURCE\_ID, RE.TITLE, RE.RESOURCE\_TYPE, RE.RESOURCE\_CLASS, CO.COPY\_NUMBER, CO.LOAN\_PERIOD, PI.PUBLISHER, PI.ISBN\_ISSN, MI.PRODUCER, LEN.LENGTH

FROM RESOURCES RE

LEFT OUTER JOIN COPY CO

ON RE.RESOURCE\_ID = CO.RESOURCE\_ID

FULL OUTER JOIN PRINT\_INFO PI

ON RE.RESOURCE\_ID = PI.RESOURCE\_ID

FULL OUTER JOIN MM\_INFO MI

ON RE.RESOURCE\_ID = MI.RESOURCE\_ID

FULL OUTER JOIN MM\_LENGTH LEN

ON RE.RESOURCE\_ID = LEN.RESOURCE\_ID

) C

LEFT OUTER JOIN (

SELECT RESOURCE\_CLASS, LISTAGG(SHELF\_NUMBER, '; ') WITHIN GROUP (ORDER BY SHELF\_NUMBER) "SHELVES"

FROM RESOURCE\_SHELF

GROUP BY RESOURCE\_CLASS

ORDER BY RESOURCE\_CLASS

) SH

ON C.RESOURCE\_CLASS = SH.RESOURCE\_CLASS

FULL OUTER JOIN (

SELECT RESOURCE\_ID, LISTAGG(AUTHOR, '; ') WITHIN GROUP (ORDER BY AUTHOR) "AUTHORS"

FROM PRINT\_AUTHOR

GROUP BY RESOURCE\_ID

ORDER BY RESOURCE\_ID

) AUT

ON C.RESOURCE\_ID = AUT.RESOURCE\_ID

) RE\_CO\_SH

LEFT OUTER JOIN LO

ON RE\_CO\_SH.RESOURCE\_ID = LO.RESOURCE\_ID AND RE\_CO\_SH.COPY\_NUMBER = LO.COPY\_NUMBER

ORDER BY RE\_CO\_SH.RESOURCE\_TYPE, RE\_CO\_SH.RESOURCE\_ID, RE\_CO\_SH.COPY\_NUMBER;

The output obtained is as follow:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RESOURCE\_ID** | **TITLE** | **RESOURCE\_TYPE** | **RESOURCE\_CLASS** | **COPY\_NUMBER** | **LOAN\_PERIOD** | **SHELVES** | **PUBLISHER** | **ISBN\_ISSN** | **AUTHORS** | **PRODUCER** | **LENGTH** | **LOANED** | **OVERDUE** | **DUE\_DATE** |
| 13951381 | Huge Dorrit | Book | Classic Fiction | 1 | 14 | 1004; 1011; 1024 | Alaska Register | 5748164967101 | T. Adams | - | - | - | - | - |
| 13951381 | Huge Dorrit | Book | Classic Fiction | 2 | 14 | 1004; 1011; 1024 | Alaska Register | 5748164967101 | T. Adams | - | - | - | - | - |
| 13951381 | Huge Dorrit | Book | Classic Fiction | 3 | 14 | 1004; 1011; 1024 | Alaska Register | 5748164967101 | T. Adams | - | - | - | - | - |
| 13951381 | Huge Dorrit | Book | Classic Fiction | 4 | 14 | 1004; 1011; 1024 | Alaska Register | 5748164967101 | T. Adams | - | - | - | - | - |
| 17649202 | The Principia: Mathematical Principles of Natural Philosophy | Book | Math | 1 | 14 | 2057; 2092 | Edi Multimedia | 9541847279186 | L. Tolstoy | - | - | YES | - | 18-DEC-21 |
| 18736414 | Jim Henson: The Biography | Book | Biography | 1 | 14 | 2026 | Sidney Herald | 2312083128058 | L. Tolstoy; M. Hill | - | - | - | - | - |
| 18736414 | Jim Henson: The Biography | Book | Biography | 2 | 14 | 2026 | Sidney Herald | 2312083128058 | L. Tolstoy; M. Hill | - | - | - | - | - |
| 23315921 | Macmillan English Dictionary for Advanced Learners | Book | Dictionary | 1 | 14 | - | Gross Publications | 9310831842018 | A. Hopkins | - | - | - | - | - |
| 27369096 | The Lean Startup | Book | Business/economics | 1 | 14 | 2040 | Rlg Hunting Stands | 9781234567897 | A. Hopkins; J. Wong | - | - | - | - | - |
| 35537314 | Introduction to Quantum Mechanics | Book | Science | 1 | 2 | 3051; 3073; 3074; 3077; 3084; 3095 | Chiworks | 3241624917491 | L. Tolstoy; R. Howard | - | - | - | - | - |
| 43106382 | Between the Lines | Book | Art/architecture | 1 | 2 | 2017 | Mindlab Media Inc | 5398164287101 | M. Abe | - | - | - | - | - |
| 63290045 | The Encyclopedia of Gods | Book | Encyclopedia | 1 | -1 | 1091 | Alaska Register | 3197752917420 | J. Ackerman | - | - | - | - | - |
| 82120851 | Fundamentals of Aerodynamics | Book | Science | 1 | 14 | 3051; 3073; 3074; 3077; 3084; 3095 | Smith Printing CO | 1294719419971 | L. Tolstoy; R. Tagore | - | - | - | - | - |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 1 | 14 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | - | - | - |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 10 | 2 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | - | - | - |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 11 | -1 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | - | - | - |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 2 | 14 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | - | - | - |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 3 | 14 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | YES | YES | 06-DEC-21 |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 4 | 14 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | - | - | - |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 5 | 14 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | - | - | - |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 6 | 14 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | - | - | - |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 7 | 14 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | - | - | - |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 8 | 2 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | - | - | - |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 9 | 2 | 1052; 1054; 1063 | The Atlanta Tribune | 3180538714538 | D. Bains; L. Tolstoy; L. Urban; Y. Agnew | - | - | - | - | - |
| 88147914 | The Truth about Cancer | Book | Journal | 1 | 14 | 1070; 1080 | Dominique Olbrechts | 5830813792471 | E. Dickinson | - | - | - | - | - |
| 20174519 | Helen of Troy | CD | History | - | - | - | - | - | - | Comcast | 45 | - | - | - |
| 47452487 | Fated Blades | CD | Science Fiction | 1 | 14 | 1027; 1032 | - | - | - | Hachette Livre | 52 | - | - | - |
| 48527649 | The History of God | CD | Religion | 1 | 14 | 1082 | - | - | - | Sony | 47 | - | - | - |
| 19270142 | Best Goals Scored | DVD | Sports and leisure | 1 | 14 | 1082 | - | - | - | Pan Macmillan | 126 | - | - | - |
| 66138583 | Japanese Style Gardens | DVD | Home and garden | - | - | 1082 | - | - | - | Bloomsbury | 155 | - | - | - |
| 87386855 | Jade Legacy | DVD | Fantasy | 1 | 14 | - | - | - | - | Pixar | 95 | - | - | - |
| 87386855 | Jade Legacy | DVD | Fantasy | 2 | 2 | - | - | - | - | Pixar | 95 | - | - | - |
| 09496244 | Crossfire | Video | Thriller | 1 | 14 | - | - | - | - | Universal Studios | 43 | - | - | - |
| 31512743 | Tinker Tailor Soldier Spy | Video | Crime | 1 | 14 | 1049 | - | - | - | Walt Disney | 30 | - | - | - |
| 31512743 | Tinker Tailor Soldier Spy | Video | Crime | 2 | 14 | 1049 | - | - | - | Walt Disney | 30 | - | - | - |
| 49873226 | Tackling Back Pain | Video | Health | 1 | 14 | 1082 | - | - | - | Fox | 28 | - | - | - |

**All\_loans**

The view all\_loans enables librarians to view all previous and current loans, where the items being loaned are still within the library collection.

The view utilises inner joins, case statements and subqueries to allow librarians to look for each loan record. Compared to directly querying the Loan table, this view also displays the extra information of some resource information, including the title and the resource type, as well as some copy information, including the loan period. Finally, the due date of the loan record, and whether the loan record was/ will be fined, and the corresponding fine amount are also displayed.

As such, librarians can quickly view historic and outstanding fines, as well as the frequency of resources being borrowed over time.

SQL scripts for creating the all\_loans view are as follow:

CREATE VIEW all\_loans AS

SELECT LO\_RE.LOAN\_ID, LO\_RE.LIBCARD\_ID, LO\_RE.RESOURCE\_ID, LO\_RE.RESOURCE\_TYPE, LO\_RE.TITLE, LO\_RE.COPY\_NUMBER, CO.LOAN\_PERIOD, LO\_RE.DATE\_BORROW, LO\_RE.date\_borrow + CO.loan\_period AS DATE\_DUE, LO\_RE.DATE\_RETURN,

CASE WHEN LO\_RE.date\_return IS NOT NULL THEN

CASE WHEN LO\_RE.date\_borrow + CO.loan\_period < LO\_RE.date\_return THEN FLOOR(LO\_RE.date\_return - (LO\_RE.date\_borrow + CO.loan\_period)) END

ELSE

CASE WHEN LO\_RE.date\_borrow + CO.loan\_period < SYSDATE THEN FLOOR(SYSDATE - (LO\_RE.date\_borrow + CO.loan\_period)) END

END AS FINE

FROM (

SELECT LO.LOAN\_ID, LO.RESOURCE\_ID, RE.RESOURCE\_TYPE, RE.TITLE, LO.LIBCARD\_ID, LO.COPY\_NUMBER, LO.DATE\_BORROW, LO.DATE\_RETURN

FROM LOAN LO

INNER JOIN RESOURCES RE

ON LO.RESOURCE\_ID = RE.RESOURCE\_ID) LO\_RE

INNER JOIN COPY CO

ON LO\_RE.RESOURCE\_ID = CO.RESOURCE\_ID AND LO\_RE.COPY\_NUMBER = CO.COPY\_NUMBER

ORDER BY LOAN\_ID;

Below is the output obtained:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **LOAN\_ID** | **LIBCARD\_ID** | **RESOURCE\_ID** | **RESOURCE\_TYPE** | **TITLE** | **COPY\_NUMBER** | **LOAN\_PERIOD** | **DATE\_BORROW** | **DATE\_DUE** | **DATE\_RETURN** | **FINE** |
| 1 | 5739753 | 35537314 | Book | Introduction to Quantum Mechanics | 1 | 2 | 30-AUG-21 | 01-SEP-21 | 01-SEP-21 | - |
| 2 | 2382819 | 17649202 | Book | The Principia: Mathematical Principles of Natural Philosophy | 1 | 14 | 31-AUG-21 | 14-SEP-21 | 02-SEP-21 | - |
| 3 | 3239001 | 82120851 | Book | Fundamentals of Aerodynamics | 1 | 14 | 01-SEP-21 | 15-SEP-21 | 15-SEP-21 | - |
| 4 | 3239001 | 18736414 | Book | Jim Henson: The Biography | 2 | 14 | 01-SEP-21 | 15-SEP-21 | 15-SEP-21 | - |
| 5 | 2830482 | 43106382 | Book | Between the Lines | 1 | 2 | 02-SEP-21 | 04-SEP-21 | 04-SEP-21 | - |
| 6 | 6177322 | 85815155 | Book | UNIX and Linux System Administration Handbook | 6 | 14 | 04-SEP-21 | 18-SEP-21 | 20-SEP-21 | 2 |
| 7 | 9538839 | 85815155 | Book | UNIX and Linux System Administration Handbook | 3 | 14 | 05-SEP-21 | 19-SEP-21 | 07-SEP-21 | - |
| 8 | 9538839 | 13951381 | Book | Huge Dorrit | 2 | 14 | 06-OCT-21 | 20-OCT-21 | 20-OCT-21 | - |
| 9 | 8592723 | 13951381 | Book | Huge Dorrit | 4 | 14 | 08-OCT-21 | 22-OCT-21 | 22-OCT-21 | - |
| 10 | 8083280 | 87386855 | DVD | Jade Legacy | 1 | 14 | 15-NOV-21 | 29-NOV-21 | 08-DEC-21 | 9 |
| 11 | 5739753 | 85815155 | Book | UNIX and Linux System Administration Handbook | 3 | 14 | 22-NOV-21 | 06-DEC-21 | - | 10 |
| 12 | 8592723 | 17649202 | Book | The Principia: Mathematical Principles of Natural Philosophy | 1 | 14 | 04-DEC-21 | 18-DEC-21 | - | - |

**All\_users**

The third and final view is all\_users. From this view, librarians can view all personal particulars of all users. When comparing to directly querying the Users table, three extra columns are shown, which are the borrowing limit of the user, the number of items currently being borrowed by the user, and the total outstanding fine, if any.

SQL scripts of the all\_users view are as follow:

CREATE VIEW all\_users AS

WITH LO AS (

SELECT LIBCARD\_ID, COUNT(LIBCARD\_ID) CURRENT\_LOAN, SUM(FINE) TOTAL\_OS\_FINE

FROM ALL\_LOANS

WHERE DATE\_RETURN IS NULL

GROUP BY LIBCARD\_ID, FINE)

SELECT U.LIBCARD\_ID, U.LAST\_NAME, U.FIRST\_NAME, U.COUNTRY\_CODE, U.MOBILE\_NUMBER, U.USER\_TYPE, UL.BORROWING\_LIMIT, U.USER\_STATUS, LO.CURRENT\_LOAN, LO.TOTAL\_OS\_FINE

FROM LO

RIGHT OUTER JOIN USERS U

ON LO.LIBCARD\_ID = U.LIBCARD\_ID

RIGHT OUTER JOIN USER\_LIMIT UL

ON U.USER\_TYPE = UL.USER\_TYPE

ORDER BY LIBCARD\_ID;

Below output of all\_users can be obtained:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **LIBCARD\_ID** | **LAST\_NAME** | **FIRST\_NAME** | **COUNTRY\_CODE** | **MOBILE\_NUMBER** | **USER\_TYPE** | **BORROWING\_LIMIT** | **USER\_STATUS** | **CURRENT\_LOAN** | **TOTAL\_OS\_FINE** |
| 2382819 | Jemmy | Wash | 44 | 3262828952 | Staff | 10 | Active | - | - |
| 2830482 | Thibaut | Thorold | 44 | 62051043520 | Student | 5 | Active | - | - |
| 3239001 | Bealle | Diperaus | 44 | 50533368146 | Student | 5 | Deactivated | - | - |
| 4282801 | Latisha | Pearson | 44 | 49501947110 | Student | 5 | Suspended | - | - |
| 5739753 | Michaella | Strafen | 353 | 14370969 | Student | 5 | Active | 1 | 10 |
| 6177322 | Barbie | Lukasik | 358 | 508381212 | Staff | 10 | Active | - | - |
| 7171742 | Bary | Medley | 44 | 76023729185 | Staff | 10 | Active | - | - |
| 8083280 | Thorsten | Glenn | 44 | 58788472692 | Student | 5 | Active | - | - |
| 8592723 | Theodora | Kantor | 44 | 27777068036 | Student | 5 | Active | 1 | - |
| 9538839 | Kemmis | Loadin | 44 | 12900789282 | Student | 5 | Active | - | - |

**Queries**

12 different queries were written to obtain information from the database, to update the tables within the database, and to add new records.

In this section, explanation of what the query is intended to do, the SQL script, and the output are included as required.

**Query 1 - Find the type, the genre, how many copies of it are held by the library and how many are loaned out**

This query utilises joins, first by left outer joining the Resources table with the Copy table to get the resource ID, resource type, resource class and the count of copies available for each resource, followed by outer joining the Loan table to see how many of these copies are currently being loaned out. To achieve the counting function, GROUP BY functions are also used. SQL scripts below:

SELECT R.RESOURCE\_ID, R.RESOURCE\_TYPE, R.RESOURCE\_CLASS, R.COPIES, LOANCOUNT.LOANED

FROM (

SELECT RE.RESOURCE\_ID, RE.RESOURCE\_TYPE, RE.RESOURCE\_CLASS, CO.COPIES

FROM RESOURCES RE

LEFT OUTER JOIN (SELECT RESOURCE\_ID, COUNT(COPY\_NUMBER) AS COPIES

FROM COPY

GROUP BY RESOURCE\_ID) CO

ON RE.RESOURCE\_ID = CO.RESOURCE\_ID

) R

FULL OUTER JOIN (

SELECT RESOURCE\_ID, COUNT(RESOURCE\_ID) AS LOANED

FROM LOAN

WHERE DATE\_RETURN IS NULL

GROUP BY RESOURCE\_ID

) LOANCOUNT

ON R.RESOURCE\_ID = LOANCOUNT.RESOURCE\_ID;

Corresponding output as follow:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RESOURCE\_ID** | **RESOURCE\_TYPE** | **RESOURCE\_CLASS** | **COPIES** | **LOANED** |
| 09496244 | Video | Thriller | 1 | - |
| 13951381 | Book | Classic Fiction | 4 | - |
| 17649202 | Book | Math | 1 | 1 |
| 47452487 | CD | Science Fiction | 1 | - |
| 85815155 | Book | Textbook | 11 | 1 |
| 49873226 | Video | Health | 1 | - |
| 63290045 | Book | Encyclopedia | 1 | - |
| 48527649 | CD | Religion | 1 | - |
| 88147914 | Book | Journal | 1 | - |
| 27369096 | Book | Business/economics | 1 | - |
| 23315921 | Book | Dictionary | 1 | - |
| 18736414 | Book | Biography | 2 | - |
| 19270142 | DVD | Sports and leisure | 1 | - |
| 31512743 | Video | Crime | 2 | - |
| 87386855 | DVD | Fantasy | 2 | - |
| 35537314 | Book | Science | 1 | - |
| 43106382 | Book | Art/architecture | 1 | - |
| 82120851 | Book | Science | 1 | - |
| 66138583 | DVD | Home and garden | - | - |
| 20174519 | CD | History | - | - |

**Query 2 - See the possible shelf location of all resources, also showing the resource title, type and genre**

While this query can be easily achieved by joining the Resource table with the Resource\_shelf table; however, this will result in one resource appearing in multiple rows when the genre it belongs to is large and spans across more than one shelves.

To tackle this issue, the LISTAGG function is used to concatenate multiple outputs of shelves into one row. The output is then ordered by Resource\_type then by Resource\_class. SQL scripts as shown below:

-- If one genre returns multiple shelves, show all shelves within one row

SELECT RE.RESOURCE\_ID, RE.TITLE, RE.RESOURCE\_TYPE, RE.RESOURCE\_CLASS, SH.SHELVES

FROM RESOURCES RE

INNER JOIN (

SELECT RESOURCE\_CLASS, LISTAGG(SHELF\_NUMBER, '; ') WITHIN GROUP (ORDER BY SHELF\_NUMBER) "SHELVES"

FROM RESOURCE\_SHELF

GROUP BY RESOURCE\_CLASS

ORDER BY RESOURCE\_CLASS

) SH

ON RE.RESOURCE\_CLASS = SH.RESOURCE\_CLASS

ORDER BY RESOURCE\_TYPE ASC, RESOURCE\_CLASS ASC;

Output obtained as follow:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RESOURCE\_ID** | **TITLE** | **RESOURCE\_TYPE** | **RESOURCE\_CLASS** | **SHELVES** |
| 43106382 | Between the Lines | Book | Art/architecture | 2017 |
| 18736414 | Jim Henson: The Biography | Book | Biography | 2026 |
| 27369096 | The Lean Startup | Book | Business/economics | 2040 |
| 13951381 | Huge Dorrit | Book | Classic Fiction | 1004; 1011; 1024 |
| 63290045 | The Encyclopedia of Gods | Book | Encyclopedia | 1091 |
| 88147914 | The Truth about Cancer | Book | Journal | 1070; 1080 |
| 17649202 | The Principia: Mathematical Principles of Natural Philosophy | Book | Math | 2057; 2092 |
| 35537314 | Introduction to Quantum Mechanics | Book | Science | 3051; 3073; 3074; 3077; 3084; 3095 |
| 82120851 | Fundamentals of Aerodynamics | Book | Science | 3051; 3073; 3074; 3077; 3084; 3095 |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 1052; 1054; 1063 |
| 48527649 | The History of God | CD | Religion | 1082 |
| 47452487 | Fated Blades | CD | Science Fiction | 1027; 1032 |
| 66138583 | Japanese Style Gardens | DVD | Home and garden | 1082 |
| 19270142 | Best Goals Scored | DVD | Sports and leisure | 1082 |
| 31512743 | Tinker Tailor Soldier Spy | Video | Crime | 1049 |
| 49873226 | Tackling Back Pain | Video | Health | 1082 |

**Query 3 – Find resources with no copies**

For the third query, we look for resources with no copies. In the context of the library, when a copy is lost, or destroyed, they might be removed from the Copy table. In case of that being the last copy of a resource, a resource might have no copies left, and the library would have to decide whether to delete the resource altogether, or to replenish their collections. To do such a query, we make use of the NOT EXISTS function, finding a resource ID which does not appear in the Copy table, where the resource ID is part of the composite primary key. SQL scripts as follows:

SELECT \*

FROM RESOURCES RE

WHERE NOT EXISTS (

SELECT \*

FROM COPY

WHERE RE.RESOURCE\_ID = COPY.RESOURCE\_ID);

And the obtained output looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| **RESOURCE\_ID** | **RESOURCE\_TYPE** | **RESOURCE\_CLASS** | **TITLE** |
| 66138583 | DVD | Home and garden | Japanese Style Gardens |
| 20174519 | CD | History | Helen of Troy |

**Query 4 – Add loan record when a copy is being borrowed**

We then move on to some more functional queries. When an item is being borrowed, a new tuple is added to the Loan table. For instance, if copy #1 of the resource #85815155 is being borrowed by user #4282801, the SQL scripts would look like this:

INSERT INTO Loan (libcard\_id, resource\_id, copy\_number, date\_borrow) VALUES ('4282801','85815155','1',TO\_DATE(SYSDATE));

Apart from entering the mentioned fields, the date\_borrow would have the value of SYSDATE. Leaving the loan\_id field out, the system would automatically generate the key. We can make a query on the Loan table by using this script:

SELECT \* FROM Loan WHERE Libcard\_ID = '4282801'AND Resource\_ID = '85815155';

The following output is returned:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LOAN\_ID** | **LIBCARD\_ID** | **RESOURCE\_ID** | **COPY\_NUMBER** | **DATE\_BORROW** | **DATE\_RETURN** |
| 13 | 4282801 | 85815155 | 1 | 16-DEC-21 | - |

**Query 5 – Modify loan record when a copy is being returned**

When a copy is being returned, we have to update the row which was created with the copy being borrowed. We assume the process is done by matching the row in the loan table where the resource ID and copy number match with that of the item being returned. There should only be one such record with the return date being a NULL value as a copy cannot be borrowed for more than once without returning. We then set the return date to SYSDATE. If the item being returned is copy #1 of resource #17649202, SQL code will be as follows:

UPDATE Loan

SET date\_return = SYSDATE

WHERE date\_return IS NULL AND copy\_number = '1' AND resource\_id = '17649202';

We can now query the Loan table and see if the return of item has been recorded properly.

SELECT \* FROM Loan WHERE copy\_number = '1' AND resource\_id = '17649202';

And we can retrieve the following two records, with the bottom one of loan\_id #12 is the record of item being returned today (16 Dec).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LOAN\_ID** | **LIBCARD\_ID** | **RESOURCE\_ID** | **COPY\_NUMBER** | **DATE\_BORROW** | **DATE\_RETURN** |
| 2 | 2382819 | 17649202 | 1 | 31-AUG-21 | 02-SEP-21 |
| 12 | 8592723 | 17649202 | 1 | 04-DEC-21 | 16-DEC-21 |

**Query 6 - Find the top three most borrowed resources**

One of the reasons of recording all the loan record history is to look for resources that are popular. In this query we will make use of the GROUP BY function to query the Loan table, order the resources by their count of records in descending order, and finally use the ROWNUM function to get the top three most borrowed resources. SQL scripts are as follows:

SELECT Re.Resource\_ID, Re.Title, Re.Resource\_type, Re.Resource\_class, Freq.Frequency

FROM (

SELECT Resource\_ID, count(Resource\_ID) as Frequency

FROM Loan

GROUP BY Resource\_ID

ORDER BY Frequency Desc

) Freq

INNER JOIN RESOURCES Re

ON Freq.Resource\_ID = Re.Resource\_ID

WHERE ROWNUM <= 3;

This is the results obtained:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RESOURCE\_ID** | **TITLE** | **RESOURCE\_TYPE** | **RESOURCE\_CLASS** | **FREQUENCY** |
| 85815155 | UNIX and Linux System Administration Handbook | Book | Textbook | 4 |
| 17649202 | The Principia: Mathematical Principles of Natural Philosophy | Book | Math | 2 |
| 13951381 | Huge Dorrit | Book | Classic Fiction | 2 |

I am not entirely surprised a textbook topped the chart at a college library.

**Query 7 - Find resources borrowed in November that have not been borrowed before during the year**

In this query we make use of the MINUS function to take away a part of results from another query. EXTRACT is also used to identify resources being borrowed in a given month from the date\_borrow of records in the Loan table. SQL codes below:

SELECT Nov.Resource\_ID, RE.Title, RE.Resource\_type, RE.Resource\_class

FROM (

SELECT Resource\_ID FROM Loan

WHERE EXTRACT(MONTH FROM date\_borrow) = 11 AND EXTRACT(YEAR FROM date\_borrow) = 2021

MINUS

SELECT Resource\_ID FROM Loan

WHERE EXTRACT(MONTH FROM date\_borrow) < 11) Nov AND EXTRACT(YEAR FROM date\_borrow) = 2021

INNER JOIN Resources RE

ON Nov.Resource\_ID = RE.Resource\_ID;

Which gave us the output:

|  |  |  |  |
| --- | --- | --- | --- |
| **RESOURCE\_ID** | **TITLE** | **RESOURCE\_TYPE** | **RESOURCE\_CLASS** |
| 87386855 | Jade Legacy | DVD | Fantasy |

**Query 8 - Find the month with the most borrowings and display the number of borrowings in the month**

In this query we are going to find the busiest month of the library and to find out how many loan records are there in the month. To achieve this, we first have to use two group loan records in the Loan table by the borrowing date as the subquery, then issue another query, doing the exact same GROUP BY method, HAVING the count of record equal that of the maximum found in the subquery. SQL scripts are shown below:

SELECT EXTRACT(MONTH FROM date\_borrow) MONTH, COUNT(Loan\_ID) "Number of borrowings"

FROM Loan

GROUP BY EXTRACT(MONTH FROM date\_borrow)

HAVING COUNT(Loan\_ID) = (

SELECT MAX(COUNT(Loan\_ID))

FROM Loan

GROUP BY EXTRACT(MONTH FROM date\_borrow)

);

Giving us the following output:

|  |  |
| --- | --- |
| **MONTH** | **Number of borrowings** |
| 9 | 5 |

**Query 9 - Find the book(s) jointly authored by L. Tolstoy and M. Hill, its genre, located shelf and floor**

For query 9, it is more of a search function conducted by the users of the library. As we have a separate Print\_author table, we can use the INTERSECT method to find out resource IDs matching both authors, then we can join the Resource, Resource\_shelf and Location tables in order to also display the genre, located shelf, and the floor number. Below are the SQL scripts:

SELECT R.RESOURCE\_ID, R.RESOURCE\_TYPE, R.RESOURCE\_CLASS, R.SHELF\_NUMBER, LO.FLOOR\_NUMBER

FROM(

SELECT RE.RESOURCE\_ID, RE.RESOURCE\_TYPE, RE.RESOURCE\_CLASS, SH.SHELF\_NUMBER

FROM RESOURCES RE

INNER JOIN RESOURCE\_SHELF SH

ON RE.RESOURCE\_CLASS = SH.RESOURCE\_CLASS

WHERE RESOURCE\_ID =

(

SELECT RESOURCE\_ID

FROM PRINT\_AUTHOR

WHERE AUTHOR = 'L. Tolstoy'

INTERSECT

SELECT RESOURCE\_ID

FROM PRINT\_AUTHOR

WHERE AUTHOR = 'M. Hill'

)

) R

INNER JOIN LOCATION LO

ON R.SHELF\_NUMBER = LO.SHELF\_NUMBER;

And the output is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RESOURCE\_ID** | **RESOURCE\_TYPE** | **RESOURCE\_CLASS** | **SHELF\_NUMBER** | **FLOOR\_NUMBER** |
| 18736414 | Book | Biography | 2026 | 2 |

**Query 10 - Find the resource type with the shortest average length and display the average length**

This query joins the Resource table with the MM\_Length table, then uses GROUP BY to find out the average length of each resource type. Using the above as a subquery, we can find out the resource type with the shortest average length by using the MIN operator. We also correct the average length to the nearest 2 decimal places by the TO\_CHAR method. SQL code below:

WITH TYPES\_LENGTH AS (

SELECT RESOURCE\_TYPE, AVG(LENGTH) AS AVERAGE\_LENGTH

FROM (

SELECT RE.RESOURCE\_ID, RE.RESOURCE\_TYPE, LEN.LENGTH

FROM RESOURCES RE

INNER JOIN MM\_LENGTH LEN

ON RE.RESOURCE\_ID = LEN.RESOURCE\_ID

)

GROUP BY RESOURCE\_TYPE

)

SELECT RESOURCE\_TYPE, TO\_CHAR(AVERAGE\_LENGTH, 'FM9999999.90') AS AVERAGE\_LENGTH

FROM TYPES\_LENGTH

WHERE AVERAGE\_LENGTH = (SELECT MIN(AVERAGE\_LENGTH) FROM TYPES\_LENGTH);

The corresponding output is as follow:

|  |  |
| --- | --- |
| **RESOURCE\_TYPE** | **AVERAGE\_LENGTH** |
| Video | 33.67 |

**Query 11 - Find the genres with no shelves and shelves with no resources**

This query will be useful when librarians would like to re-organise the shelves and the genres. Using full outer join on the resource\_class, we can find out which resource\_class has the corresponding shelf\_number value as NULL, and which shelf\_number has the corresponding resource\_class value of NULL. SQL query as follows:

SELECT RE.RESOURCE\_CLASS, SH.SHELF\_NUMBER

FROM RESOURCES RE

FULL OUTER JOIN RESOURCE\_SHELF SH

ON RE.RESOURCE\_CLASS = SH.RESOURCE\_CLASS

WHERE RE.RESOURCE\_CLASS IS NULL OR SH.SHELF\_NUMBER IS NULL;

And with the output being:

|  |  |
| --- | --- |
| **RESOURCE\_CLASS** | **SHELF\_NUMBER** |
| - | 2019 |
| - | 1094 |
| - | 2045 |
| - | 2043 |
| - | 2056 |
| - | 1049 |
| - | 1049 |
| Fantasy | - |
| Thriller | - |
| History | - |
| Dictionary | - |

Indicating there are indeed quite a number of empty shelves and a few genres with no designated shelves.

**Query 12 - Find the library card IDs of users being suspended for reasons other than currently having an overdue item**

In our assumptions of the library, some users can be suspended for reasons other than currently having an overdue item, with outstanding fines reaching $10. To find out who these users are, we can first find out the users who are suspended, then MINUS the ones who should have been suspended with having a loan record of over $10 of outstanding fines. SQL codes as follow:

SELECT LIBCARD\_ID

FROM USERS

WHERE USER\_STATUS = 'Suspended'

MINUS

SELECT LIBCARD\_ID

FROM LOAN LO

INNER JOIN COPY CO

ON LO.RESOURCE\_ID = CO.RESOURCE\_ID AND LO.COPY\_NUMBER = CO.COPY\_NUMBER

WHERE DATE\_RETURN IS NULL AND FLOOR(SYSDATE - (LO.date\_borrow + CO.loan\_period)) >= 10;

Noting that we have used the FLOOR function here, since the SYSDATE element in Oracle being so precise that, without using the FLOOR function, the fine will be increased by $1/24 for every hour, and $1/1440 for every minute past the due date.

The output of the query is as follows:

|  |
| --- |
| **LIBCARD\_ID** |
| 4282801 |

**Creating Triggers**

Four triggers are created for the database. These triggers are created for the enforcement of business rules, notification of the occurrence certain conditions, and auditing purposes.

**Trigger 1 - Stop users from borrowing resources** in the following situations:

* The borrowing limit is reached for the user
* The resource borrowed can only be used within the library
* The copy has previously been borrowed but has not been returned properly
* The user is trying to borrow the same copy for the second time within a day
* The user’s library card is suspended or deactivated

This trigger is for enforcing the business rules of the library, and in creating this trigger, we try to declare multiple variables which reflect the status of a user, a copy or of a loan record, and if these variables match certain conditions, the addition of the loan record will be aborted before execution.

The code of the trigger is as follows:

CREATE OR REPLACE TRIGGER abort\_loan

BEFORE INSERT ON LOAN

FOR EACH ROW

DECLARE

lim User\_limit.borrowing\_limit%TYPE;

curr\_loan NUMBER(2);

period NUMBER(2);

loaned NUMBER(1);

same\_day NUMBER(1);

status Users.user\_status%TYPE;

BEGIN

SELECT borrowing\_limit

INTO lim

FROM USERS U

INNER JOIN user\_limit ul

ON U.user\_type = ul.user\_type

WHERE libcard\_ID = :NEW.libcard\_ID;

SELECT loan\_period

INTO period

FROM copy

WHERE resource\_id = :NEW.resource\_id AND copy\_number = :NEW.copy\_number;

BEGIN

SELECT COUNT(libcard\_id)

INTO curr\_loan

FROM loan

WHERE date\_return IS NULL AND libcard\_ID = :NEW.libcard\_ID

GROUP BY libcard\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

curr\_loan := 0;

END;

BEGIN

SELECT COUNT(loan\_id)

INTO loaned

FROM loan

WHERE date\_return IS NULL AND resource\_id = :NEW.resource\_id AND copy\_number = :NEW.copy\_number

GROUP BY loan\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

loaned := 0;

END;

BEGIN

SELECT COUNT(loan\_id)

INTO same\_day

FROM loan

WHERE date\_borrow = :NEW.date\_borrow AND resource\_id = :NEW.resource\_id AND copy\_number = :NEW.copy\_number AND libcard\_ID = :NEW.libcard\_ID

GROUP BY loan\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

same\_day := 0;

END;

SELECT user\_status

INTO status

FROM USERS

WHERE libcard\_ID = :NEW.libcard\_ID;

IF curr\_loan >= lim THEN

raise\_application\_error(-20001, 'Borrowing limit is exceeded.');

ELSIF period = -1 THEN

raise\_application\_error(-20001, 'The copy is only allowed to be used within the library');

ELSIF loaned = 1 THEN

raise\_application\_error(-20001, 'The copy has not been returned properly.');

ELSIF same\_day = 1 THEN

raise\_application\_error(-20001, 'The copy had been borrowed by the same user today.');

ELSIF status <> 'Active' THEN

raise\_application\_error(-20001, 'The user is not in active status.');

END IF;

END;

/

This rather long trigger can be broken down into several parts. First, the DECLARE part declares 6 variables to be used in the trigger, with *lim* reflecting the borrowing limit of the user, *curr\_loan* reflecting the current number of items borrowed by the user, *period* reflecting the maximum loan period of the copy, *same\_day* reflecting whether the user has already borrowed the same copy and returned within the same day, and *status* reflecting whether the user is in an active status.

We then assign these variables to appropriate values by using a series of SELECT INTO statements, and EXCEPTION statements to set the variable to value 0 when no data is retrieved from the SELECT query.

Afterwards, a series of IF ELSIF statements are used to raise application error when the conditions are met, rendering the addition of tuple to abort, which also warns the librarian not to loan out the resource copy.

We can test the trigger by simulating borrowing of different items by different users.

First we are going to have user #2830482 to borrow over their limit of 5 items.

INSERT INTO Loan (libcard\_id,resource\_id,copy\_number,date\_borrow) VALUES ('2830482','85815155','4',TO\_DATE(SYSDATE));

INSERT INTO Loan (libcard\_id,resource\_id,copy\_number,date\_borrow) VALUES ('2830482','85815155','5',TO\_DATE(SYSDATE));

INSERT INTO Loan (libcard\_id,resource\_id,copy\_number,date\_borrow) VALUES ('2830482','85815155','6',TO\_DATE(SYSDATE));

INSERT INTO Loan (libcard\_id,resource\_id,copy\_number,date\_borrow) VALUES ('2830482','85815155','7',TO\_DATE(SYSDATE));

INSERT INTO Loan (libcard\_id,resource\_id,copy\_number,date\_borrow) VALUES ('2830482','85815155','8',TO\_DATE(SYSDATE));

INSERT INTO Loan (libcard\_id,resource\_id,copy\_number,date\_borrow) VALUES ('2830482','85815155','9',TO\_DATE(SYSDATE));

At the sixth input, the system returns ORA-20001: Borrowing limit is exceeded.

Next, we are going to have another user #7171742 borrow copy #1 of resource #63290045, which has a loan\_period of -1, indicating that it cannot be borrowed.

INSERT INTO Loan (libcard\_id,resource\_id,copy\_number,date\_borrow) VALUES ('7171742','63290045','1',TO\_DATE(SYSDATE));

System returns ORA-20001: The copy is only allowed to be used within the library

We are then going to have the same user borrows copy #1 of the resource #85815155 is being borrowed by user #4282801, which is supposed to be loaned to another user earlier while demonstrating Query 4 above.

INSERT INTO Loan (libcard\_id, resource\_id, copy\_number, date\_borrow) VALUES ('7171742','85815155','1',TO\_DATE(SYSDATE));

System returns ORA-20001: The copy has not been returned properly.

Let us have the user #4282801 in Query 4 to return the copy immediately and borrow it again.

UPDATE Loan

SET date\_return = SYSDATE

WHERE date\_return IS NULL AND copy\_number = '1' AND resource\_id = '85815155';

INSERT INTO Loan (libcard\_id, resource\_id, copy\_number, date\_borrow) VALUES ('4282801','85815155','1',TO\_DATE(SYSDATE));

System returns:

1 row(s) updated.

ORA-20001: The copy had been borrowed by the same user today.

Which means the resource is returned but cannot be borrowed again immediately.

Finally, we will have a deactivated user # 3239001 to borrow that book.

INSERT INTO Loan (libcard\_id, resource\_id, copy\_number, date\_borrow) VALUES ('3239001','85815155','1',TO\_DATE(SYSDATE));

System returns ORA-20001: The user is not in active status.

Checking the Loan table again, except that the copy being borrowed in Query 4 is returned, and the first five borrowings of user #2830482 in the first testing were added to the table, all other records have been blocked by the trigger and have not entered the Loan table.

**Trigger 2 - Disable the update of loan table updating beyond opening hours of the library**

In ideal circumstances, database maintenance would be carried out after the opening hours of the library, where there is minimum activity in the database. To further minimise activities beyond the opening hours to facilitate system maintenance, we are creating this statement level trigger to stop all inserting and updating to the loan table, where the majority of system activities take place, outside the operating hours of 0900 to 1900. If any updating or inserting is attempted, an application error would be raised.

SQL code as follow:

CREATE OR REPLACE TRIGGER loan\_noot

BEFORE INSERT OR UPDATE

ON loan

DECLARE

l\_hour NUMBER;

BEGIN

l\_hour := EXTRACT(HOUR FROM SYSTIMESTAMP);

IF l\_hour NOT BETWEEN 09 AND 19 THEN

raise\_application\_error(-20100,'Loan records cannot be altered beyond service hours.');

END IF;

END;

/

At the time of writing this paragraph, it is 23:00. I will now have a user trying to borrow a copy and returning a copy.

INSERT INTO Loan (libcard\_id, resource\_id, copy\_number, date\_borrow) VALUES ('2382819','43106382','1',TO\_DATE(SYSDATE));

UPDATE Loan

SET date\_return = SYSDATE

WHERE date\_return IS NULL AND copy\_number = '6' AND resource\_id = '85815155';

The system returns twice the message ORA-20100: Loan records cannot be altered beyond service hours., indicating the records cannot be added and updated.

**Trigger 3 - Warn librarian when an overdue item is returned, and audit fine payments by writing to the payment table the loan\_id and payment value**

This trigger activates whenever an overdue item is being returned, the system will display a message, reminding the librarian to collect fines for the overdue loan. A new row will then automatically be inserted to the Payment table with the corresponding Loan ID and the amount fined. SQL code as follows:

CREATE OR REPLACE TRIGGER archive\_fine

FOR UPDATE OR INSERT ON loan

COMPOUND TRIGGER

TYPE r\_loan\_type IS RECORD (

loan\_id loan.loan\_id%TYPE,

resource\_id loan.resource\_id%TYPE,

copy\_number loan.copy\_number%TYPE,

date\_borrow loan.date\_borrow%TYPE,

date\_return loan.date\_return%TYPE

);

TYPE t\_loan\_type IS TABLE OF r\_loan\_type

INDEX BY PLS\_INTEGER;

t\_loan t\_loan\_type;

AFTER EACH ROW IS

BEGIN

t\_loan (t\_loan.COUNT + 1).loan\_id := :OLD.loan\_id;

t\_loan (t\_loan.COUNT).resource\_id := :OLD.resource\_id;

t\_loan (t\_loan.COUNT).copy\_number := :OLD.copy\_number;

t\_loan (t\_loan.COUNT).date\_borrow := :OLD.date\_borrow;

t\_loan (t\_loan.COUNT).date\_return := :OLD.date\_return;

END AFTER EACH ROW;

AFTER STATEMENT IS

amnt payment.payment\_amount%TYPE;

BEGIN

FOR indx IN 1 .. t\_loan.COUNT

LOOP

SELECT FINE

INTO amnt

FROM all\_loans

WHERE loan\_id = t\_loan(indx).loan\_id;

IF amnt IS NOT NULL

THEN

INSERT INTO payment VALUES (t\_loan(indx).loan\_id, amnt);

dbms\_output.put\_line( 'Please collect fine: $' || amnt || ' for this copy.' );

END IF;

END LOOP;

END AFTER STATEMENT;

END;

/

It is worth noting that for simplicity, this row level trigger will check the value of view all\_loans, which is in fact built upon the Loan table. When a trigger is activated by an update of a row in the Loan table, and then checks the Loan table itself, mutating table issue will occur. To tackle this problem, a compound trigger is used.

*“The compound trigger is introduced in Oracle Database 11g Release 1, it allows code for one or more timing points for a specific object to be combined into a single trigger.”* [[1]](#footnote-2)(Tim Hall n.d.) The compound trigger above can be broken down into three main steps. First, an array of loan record is declared with loan\_id, resource\_id, copy\_number, date\_borrow and date\_return included. Second, the affected rows are collected into the array in the row-level trigger, i.e., the AFTER EACH ROW part above. Finally, each after row is updated in the statement-level trigger, i.e., the AFTER STATEMENT part using a FOR LOOP. The use of compound trigger tackled the problem of mutating tables and avoided the use of a package.

Testing of this trigger will be done after the introduction of Trigger 4 as they are related.

**Trigger 4: Reactivating users who have paid their fines**

Also related to fine payment, this trigger turns the users’ status back to ‘Active’ after they clear their outstanding fine payment. SQL code of the trigger is shown below:

CREATE OR REPLACE TRIGGER Reactivate

AFTER INSERT ON Payment

FOR EACH ROW

DECLARE

usr Users.libcard\_ID%TYPE;

dum Users.libcard\_ID%TYPE;

os BOOLEAN;

BEGIN

SELECT libcard\_ID

INTO usr

FROM Loan

WHERE loan\_id = :NEW.loan\_id;

BEGIN

SELECT libcard\_ID

INTO dum

FROM all\_loans

WHERE libcard\_ID = usr AND date\_return IS NULL AND fine >= 10 AND loan\_id = :NEW.loan\_id;

os := TRUE;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

os := FALSE;

END;

IF os = FALSE THEN

UPDATE Users

SET user\_status = 'Active'

WHERE

libcard\_ID = usr;

END IF;

END;

/

In this row-level trigger, since in Trigger 3 we have set that when an overdue item is being returned and the fine payment is made, a new row will be inserted to the Payment table, we can activate this trigger when a new row is being inserted on the Payment table. This also avoids the mutating table problem since the trigger will then check the Loan table to see if this particular user has any borrowed items with an outstanding fine reaching $10. If there is no such record, they will have the status switched back to active.

Now we can test Triggers 3 and 4 together.

First, we will have to suspend any user having loan record overdue by 10 days. We intended to do it using a time-based trigger but since it is not supported in Live SQL we are now doing it manually by declaring a cursor and loop through it. This might actually be considered as our 13th Query.

DECLARE

CURSOR c\_overdue\_usr

IS

SELECT libcard\_id

FROM all\_loans

WHERE date\_return is NULL AND fine >= 10;

BEGIN

FOR overdue\_usr IN c\_overdue\_usr

LOOP

UPDATE Users

SET user\_status = 'Suspended'

WHERE libcard\_id = overdue\_usr.libcard\_id;

END LOOP;

END;

#5739753 should now be suspended, confirming by

SELECT \* FROM all\_users WHERE libcard\_ID = '5739753'; returns:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **LIBCARD\_ID** | **LAST\_NAME** | **FIRST\_NAME** | **COUNTRY\_CODE** | **MOBILE\_NUMBER** | **USER\_TYPE** | **BORROWING\_LIMIT** | **USER\_STATUS** | **CURRENT\_LOAN** | **TOTAL\_OS\_FINE** |
| 5739753 | Michaella | Strafen | 353 | 14370969 | Student | 5 | Suspended | 1 | 11 |

In the next step, we will have this user returning the overdue item, which is copy #3 of resource #85815155, this should trigger both triggers 3 and 4.

UPDATE Loan

SET date\_return = SYSDATE

WHERE date\_return IS NULL AND copy\_number = '3' AND resource\_id = '85815155';

System returns:

1 row(s) updated.  
Please collect fine: $11 for this copy.

The system successfully displays a reminder to the librarian that fine should be collected.

Checking the Payment table:

SELECT \* FROM Payment;

Output:

|  |  |
| --- | --- |
| **LOAN\_ID** | **PAYMENT\_AMOUNT** |
| 11 | 11 |
| 6 | 2 |

The record with Loan ID #11 is there, freshly added to the Payment table, the amount paid is also recorded.

We can also check if the date of return is properly recorded in the Loan table.

SELECT \* FROM Loan WHERE Loan\_ID = 11;

Now returns:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LOAN\_ID** | **LIBCARD\_ID** | **RESOURCE\_ID** | **COPY\_NUMBER** | **DATE\_BORROW** | **DATE\_RETURN** |
| 11 | 5739753 | 85815155 | 3 | 22-NOV-21 | 17-DEC-21 |

We can now check the all\_users view and see whether the status of this user is switched back to active:

SELECT \* FROM all\_users WHERE libcard\_ID = '5739753';

Returns:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **LIBCARD\_ID** | **LAST\_NAME** | **FIRST\_NAME** | **COUNTRY\_CODE** | **MOBILE\_NUMBER** | **USER\_TYPE** | **BORROWING\_LIMIT** | **USER\_STATUS** | **CURRENT\_LOAN** | **TOTAL\_OS\_FINE** |
| 5739753 | Michaella | Strafen | 353 | 14370969 | Student | 5 | Active | - | - |

Now the user is back to the active state. Triggers 3 and 4 both work.

**Data Security Issues**

Databases often contain sensitive information that shouldn’t be available for everyone. *“Database security refers to the range of tools, controls, and measures designed to establish and preserve database confidentiality, integrity, and availability.”*[[2]](#footnote-3)In a corporate environment the databases may be of strategic importance, which further indicates the importance of keeping the data secure and confidential.

The United Kingdom’s Data Protection Act of 2018 controls how personal information is handled by the government, organizations, and businesses. It sets out a range of guidelines that organizations that are storing personal data need to comply with.

It is important to implement relevant mechanics to protect the database against accidental and intentional threats. Data security issues that would need to be addressed by the administrators of the college library system:

Access control is one of the most important data security practices, which control a user’s ability to read or write information. In Oracle, a privilege allows a user to access or create a database object or to run certain DBMS utilities. To simplify and speed up account administration, the privilege can be grouped into roles and assign it to each user account.

Considering the college library system, we assume that we have users, librarians and the database administrator. The database administrator is responsible to grant users privileges that determine the operations they can perform upon data. For instance, all users are granted the privilege to view the “all\_copies” table. The librarians are split into two different tiers, junior and senior. Junior librarians are granted access to view resources, add to the loan table. The senior librarians are granted the same access privileges but include some extra privileges to deal with sensitive operations such as removing loan records, changing user status manually, and altering users table. These access control privileges ensure that the correct users have the appropriate rights within the database.

Oracle provides Discretionary Access Control (DAC) and Mandatory Access Control (MAC) in terms of access control. DAC provides high flexibility and relatively low implementation cost as users can share and configure data access parameters without administrators and hence it is easy to maintain. However, it does not provide a high level of data protection as users can share their data freely. For example, if a senior librarian shares the access rights (e.g. altering users table) with a junior librarian, there’s no guarantee that the junior librarian can do the job correctly. In addition, private confidential data that is stored in a user’s profile like id card number or banking details might be exposed which is not needed in transactions. In other words, it is difficult to enforce the principles of least privilege. Given the complexity of the college library system, DAC should be sufficient.

If the database would contain data considered as highly sensitive, classified and/or military grade it would have been necessary to implement MAC. MAC provides a higher level of security, but the implementation cost is higher, and it would be a lot more work for the DBA in terms of maintenance. In conclusion, a DAC would prove sufficient for the college library system.

**References**

Tim Hall, n.d., *Trigger Enhancements in Oracle Database 11g Release 1*, Oracle-Base, accessed 15 December 2021, <https://oracle-base.com/articles/11g/trigger-enhancements-11gr1>

Database Security, 16 December 2021, <https://www.ibm.com/topics/database-security>

1. https://oracle-base.com/articles/11g/trigger-enhancements-11gr1 [↑](#footnote-ref-2)
2. https://www.ibm.com/topics/database-security [↑](#footnote-ref-3)