A NORMALIZED RELATIONAL DATABASE IMPLEMENTATION: COLLEGE LIBRARY SYSTEM

Group 4

Sarah Iruobe

Roy Karlo Nuyda

Mingqiong Wu

Database Systems ECS740

Professor Tony Stockman

Queen Mary University of London

17 December, 2021

Outline

Relational Schema

The relational schema we have provided in this coursework is identical to the relational schema we provided in the first coursework, with the exception of **Fine Audit**, which was included to demonstrate an auditing trigger.

Member (member_ID, member_name, role, department, phone_number, email)

Role (role, borrow_limit)

Item (item_id, item_name, isbn_issn, author, publisher, publish_year, price, class_number)

Copy (Copy_ID, *Item_ID, Location_ID*, copy_type)

Copy Type (copy_type, loan_period)

Location (Location_ID, shelf_number, floor_number, building_number)

Loan (loan_id, *member_id*, *copy_id*, time_borrowed)

Fine (fine_id, *member_id*, *loan_id*, fine_amount)

Fine Audit (fine_id, fine_amount, date_deleted, deleted_by)

Create Table Script

We have included the script for creating our tables, including the constraints to perform validation checks on any incoming data.

```
CREATE TABLE MEMBER(
MEMBER_ID NUMBER(10),
MEMBER_NAME VARCHAR2(30) NOT NULL,
ROLE VARCHAR2(10) NOT NULL,
DEPARTMENT VARCHAR2(30),
PHONE_NUMBER VARCHAR2(30) UNIQUE,
EMAIL VARCHAR2(40) UNIQUE,
CONSTRAINT PK_MEMBER_ID PRIMARY KEY (MEMBER_ID)
);

CREATE TABLE ROLE(
ROLE VARCHAR2(10) PRIMARY KEY,
BORROW_LIMIT NUMBER(2) NOT NULL
);

CREATE TABLE ITEM(
ITEM_ID NUMBER(10),
ITEM_NAME VARCHAR2(50) NOT NULL,
ISBN_ISSN VARCHAR2(20) UNIQUE,
AUTHOR VARCHAR2(40),
PUBLISHER VARCHAR2(50),
PUBLISHER_YEAR NUMBER(4),
PRICE NUMBER,
CLASS_NUMBER NUMBER(3) NOT NULL,
CONSTRAINT PK_ITEM_ID PRIMARY KEY (ITEM_ID)
);

CREATE TABLE LOCATION(
LOCATION_ID NUMBER(10),
SHELF_NUMBER NUMBER NOT NULL,
FLOOR_NUMBER NUMBER(1) NOT NULL,
BUILDING_NUMBER VARCHAR2(10) NOT NULL,
CONSTRAINT CK_LOCATION_FLOOR CHECK (FLOOR_NUMBER BETWEEN 0 AND 2),
CONSTRAINT PK_LOCATION_ID PRIMARY KEY (LOCATION_ID)
);

CREATE TABLE COPY(
COPY_ID NUMBER(5),
ITEM_ID NUMBER(10) NOT NULL,
LOCATION_ID NUMBER(10) NOT NULL,
COPY_TYPE VARCHAR2(30) NOT NULL,
CONSTRAINT PK_COPY_ID PRIMARY KEY (COPY_ID),
CONSTRAINT FK_COPY_ITEM_ID FOREIGN KEY (ITEM_ID) REFERENCES ITEM
(ITEM_ID),
CONSTRAINT FK_COPY_LOCATION_ID FOREIGN KEY (LOCATION_ID) REFERENCES
LOCATION (LOCATION_ID)
);

CREATE TABLE COPY_TYPE(
COPY_TYPE VARCHAR2(30),
LOAN_PERIOD NUMBER(2) NOT NULL,
CONSTRAINT PK_COPY_TYPE PRIMARY KEY (COPY_TYPE)
);

CREATE TABLE LOAN(
LOAN_ID NUMBER,
M_ID NUMBER(10) NOT NULL,
COPY_ID NUMBER(5) NOT NULL,
TIME_BORROWED DATE NOT NULL,
CONSTRAINT PK_LOAN_ID PRIMARY KEY (LOAN_ID),
CONSTRAINT FK_LOAN_MEMBER FOREIGN KEY (M_ID) REFERENCES MEMBER
(M_ID),
CONSTRAINT FK_LOAN_COPY FOREIGN KEY (COPY_ID) REFERENCES COPY
(COPY_ID)
);

CREATE TABLE FINE(
FINE_ID NUMBER,
LOAN_ID NUMBER NOT NULL,
M_ID NUMBER(10) NOT NULL,
TOTAL_AMOUNT NUMBER NOT NULL,
CONSTRAINT PK_FINE_ID PRIMARY KEY (FINE_ID),
CONSTRAINT FK_FINE_MEMBER FOREIGN KEY (M_ID) REFERENCES MEMBER
(M_ID),
CONSTRAINT FK_FINE_LOAN FOREIGN KEY (LOAN_ID) REFERENCES LOAN
(LOAN_ID)
);

CREATE TABLE FINE_AUDIT(
   FINE_ID NUMBER(10) NOT NULL,
   FINE_AMOUNT NUMBER(10),
   DELETE_DATE DATE,
   DELETED_BY VARCHAR(10)
);
```

Test Data

We populated our database with test data, demonstrated by the insert statements below and the screenshots of the data retrieved by simple queries.

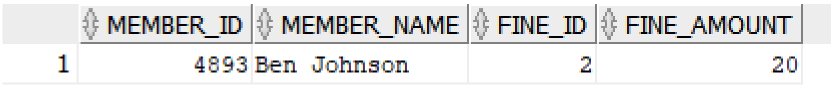
```
INSERT INTO MEMBER VALUES
(4893403432, 'Ben Johnson', 'Staff', 'Geography', 07383947294,
'ben.johnson@goldsmiths.ac.uk');
INSERT INTO MEMBER VALUES
(4879399583, 'Rebecca Moore', 'Student', 'Life Sciences', 0738379405,
'rebecca.moore@goldsmiths.ac.uk');
INSERT INTO MEMBER VALUES
(4785934850, 'Stanley Waters', 'Student', 'History', 07493759372,
'stanley.waters@goldsmiths.ac.uk');
INSERT INTO MEMBER VALUES
(47589549038, 'Joseph Smith', 'Staff', 'Economics', 18009849248 ,
'joseph.smith@goldsmiths.ac.uk');


INSERT INTO ROLE VALUES
('Staff', 10);
INSERT INTO ROLE VALUES
('Student', 5);

INSERT INTO ITEM VALUES
(8459457295,'Exploring Geography', 9843873964338, 'John Brown', 'Thompsons', 2012, 5.60,
303);
INSERT INTO ITEM VALUES
(8847304893,'Rocks and minerals', 9855875464, 'Peter Smith', 'Crown Publishers', 2005, 10.00,
303);
INSERT INTO ITEM VALUES
(5587450609, 'Proteins and peptides', '9458960956', 'Julia Peters', 'Watermark', 2008, 15.40, 312);
INSERT INTO ITEM VALUES
(6445565676,'Statistics', 9543965945343, 'Max Johnson', 'Thompsons', 2001, 20.10, 305);

INSERT INTO LOCATION VALUES
(4584737598, 12, 2, 3);
INSERT INTO LOCATION VALUES
(3345565753, 13, 0, 22);
INSERT INTO LOCATION VALUES
(5934827285, 4, 1, 5);
```

```
INSERT INTO LOCATION VALUES
(9548372685, 6, 0, 8);

INSERT INTO COPY_TYPE VALUES
('Normal', 14);
INSERT INTO COPY_TYPE VALUES
('Reference', 0);

INSERT INTO LOAN VALUES
(432, 4893403432, 389, '20-07-2021');
INSERT INTO LOAN VALUES
(894, 4879399583, 557, '04-09-2021');
INSERT INTO LOAN VALUES
(987, 47589549038, 854, '09-06-2021');
INSERT INTO LOAN VALUES
(765, 47589549038, 890, '12-03-2021');

INSERT INTO FINE VALUES
(002, 746, 4893403432, 4.00)
INSERT INTO FINE VALUES
(007, 465, 4879399583, 8.00)
INSERT INTO FINE VALUES
(003, 987, 4785934850, 3.00)
INSERT INTO FINE VALUES
(006, 817, 47589549038, 9.00)
```

Views

SUSPENDED USERS FOR DELINQUENT ACCOUNTS:
This view is specifically meant for library administrators to see a list of members with fines over the delinquent amount, in this case 10.00.

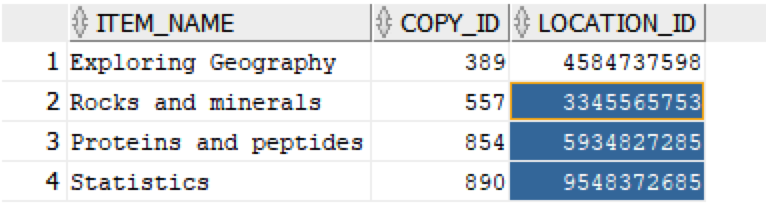
```
CREATE VIEW USERS_SUSPENDED AS SELECT
MEMBER.MEMBER_ID,MEMBER.M_NAME,FINE.FINE_ID,FINE.FINE_AMOUNT
FROM MEMBER INNER JOIN FINE ON MEMBER.MEMBER_ID=FINE.MEMBER_ID
WHERE FINE_AMOUNT>=10;
```



	MEMBER_ID	MEMBER_NAME	FINE_ID	FINE_AMOUNT
1	4893	Ben Johnson	2	20

VIEW COPIES OF AN ITEM:
This view is for librarians to see the number of copies of items in the library.

```
CREATE VIEW NUMBER_COPIES_ITEM AS
SELECT ITEM_NAME,COPY_ID,LOCATION_ID
FROM ITEM INNER JOIN COPY
ON ITEM.ITEM_ID=COPY.ITEM_ID;
```

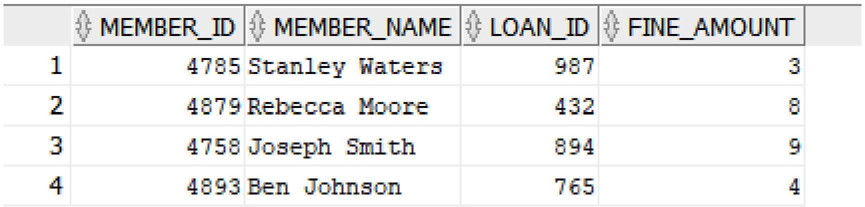


	ITEM_NAME	COPY_ID	LOCATION_ID
1	Exploring Geography	389	4584737598
2	Rocks and minerals	557	3345565753
3	Proteins and peptides	854	5934827285
4	Statistics	890	9548372685

VIEW MEMBERS WITH FINES:

This view is for library administrators to see the members with active fines, meaning fines that started to accrue.

```
CREATE VIEW MEMBER_FINES AS
SELECT MEMBER.MEMBER_ID,M_NAME,LOAN_ID,FINE_AMOUNT
FROM MEMBER INNER JOIN FINE
ON MEMBER.MEMBER_ID=FINE.MEMBER_ID
WHERE FINE_AMOUNT IS NOT NULL;
```

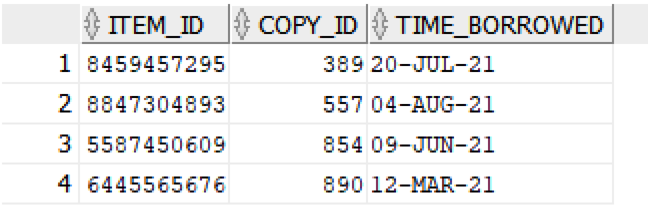


	MEMBER_ID	MEMBER_NAME	LOAN_ID	FINE_AMOUNT
1	4785	Stanley Waters	987	3
2	4879	Rebecca Moore	432	8
3	4758	Joseph Smith	894	9
4	4893	Ben Johnson	765	4

VIEW ALL CURRENT LOANS:

This view is intended for library administrators to see all the current loans active in the library.

```
CREATE VIEW ALL_CURRENT_LOAN AS
SELECT COPY.ITEM_ID, COPY.COPY_ID, LOAN.TIME_BORROWED
FROM COPY INNER JOIN LOAN
ON COPY.COPY_ID=LOAN.COPY_ID;
```

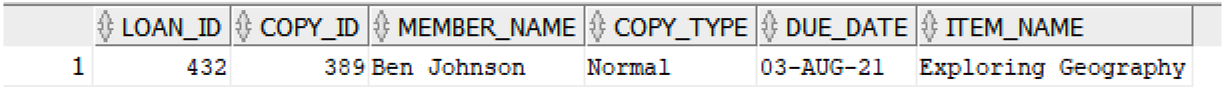


	ITEM_ID	COPY_ID	TIME_BORROWED
1	8459457295	389	20-JUL-21
2	8847304893	557	04-AUG-21
3	5587450609	854	09-JUN-21
4	6445565676	890	12-MAR-21

Queries

This query retrieves the loan information for all a member's current loans.

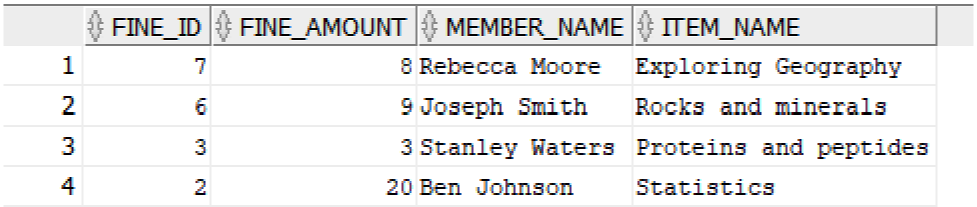
```
SELECT LOAN_ID, LOAN.COPY_ID, MEMBER.MEMBER_NAME, COPY_TYPE.COPY_TYPE, LOAN.TIME_BORROWED + COPY_TYPE.LOAN_PERIOD AS DUE_DATE, ITEM.ITEM_NAME
FROM LOAN
INNER JOIN MEMBER ON MEMBER.MEMBER_ID = LOAN.MEMBER_ID
INNER JOIN COPY ON COPY.COPY_ID = LOAN.COPY_ID
INNER JOIN ITEM ON ITEM.ITEM_ID = COPY.ITEM_ID
INNER JOIN COPY_TYPE ON COPY.COPY_TYPE = COPY_TYPE.COPY_TYPE
WHERE MEMBER.MEMBER_ID = 4893;
```



	LOAN_ID	COPY_ID	MEMBER_NAME	COPY_TYPE	DUE_DATE	ITEM_NAME
1	432	389	Ben Johnson	Normal	03-AUG-21	Exploring Geography

This query retrieves the fine information for a member's current fines.

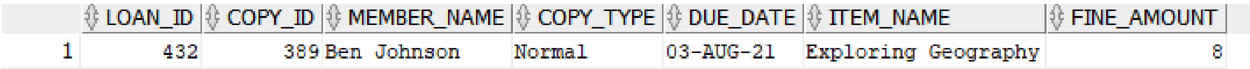
```
SELECT FINE_ID, FINE_AMOUNT, MEMBER.MEMBER_NAME, ITEM.ITEM_NAME
FROM FINE
INNER JOIN MEMBER ON MEMBER.MEMBER_ID = FINE.MEMBER_ID
INNER JOIN LOAN ON LOAN.LOAN_ID = FINE.LOAN_ID
INNER JOIN COPY ON COPY.COPY_ID = LOAN.COPY_ID
INNER JOIN ITEM ON ITEM.ITEM_ID = COPY.ITEM_ID
```



	FINE_ID	FINE_AMOUNT	MEMBER_NAME	ITEM_NAME
1	7	8	Rebecca Moore	Exploring Geography
2	6	9	Joseph Smith	Rocks and minerals
3	3	3	Stanley Waters	Proteins and peptides
4	2	20	Ben Johnson	Statistics

This query retrieves the account information for a member's current account, which is defined as both their loans and fines.

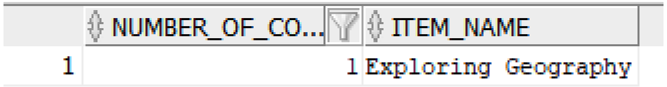
```
SELECT
LOAN.LOAN_ID, LOAN.COPY_ID, MEMBER.MEMBER_NAME,
COPY_TYPE.COPY_TYPE,
LOAN.TIME_BORROWED + COPY_TYPE.LOAN_PERIOD AS DUE_DATE,
ITEM.ITEM_NAME,
FINE.FINE_AMOUNT
FROM LOAN
INNER JOIN MEMBER ON MEMBER.MEMBER_ID = LOAN.MEMBER_ID
INNER JOIN FINE ON FINE.LOAN_ID = LOAN.LOAN_ID
INNER JOIN COPY ON COPY.COPY_ID = LOAN.COPY_ID
INNER JOIN ITEM ON ITEM.ITEM_ID = COPY.ITEM_ID
INNER JOIN COPY_TYPE ON COPY.COPY_TYPE = COPY_TYPE.COPY_TYPE
WHERE MEMBER.MEMBER_ID = 4893;
```



	LOAN_ID	COPY_ID	MEMBER_NAME	COPY_TYPE	DUE_DATE	ITEM_NAME	FINE_AMOUNT
1	432	389	Ben Johnson	Normal	03-AUG-21	Exploring Geography	8

This query counts the number of copies of an item.

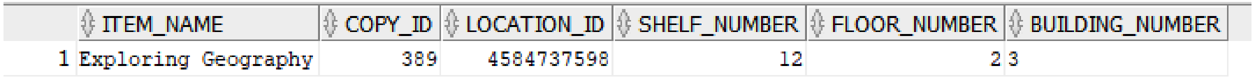
```
SELECT COUNT(COPY.ITEM_ID) AS NUMBER_OF_COPIES, ITEM.ITEM_NAME
FROM COPY
FULL OUTER JOIN ITEM ON COPY.ITEM_ID = ITEM.ITEM_ID
WHERE ITEM.ITEM_ID = 8459457295
GROUP BY ITEM.ITEM_NAME, COPY.ITEM_ID;
```



	NUMBER_OF_CO...	ITEM_NAME
1	1	Exploring Geography

This query displays all the locations for a particular item.

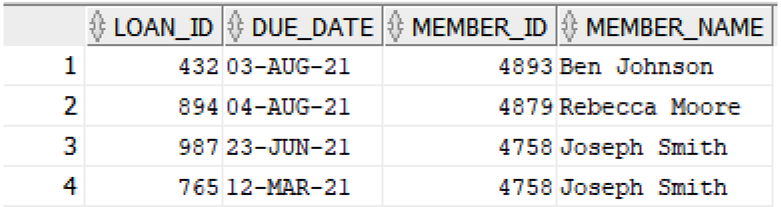
```
SELECT ITEM.ITEM_NAME, COPY.COPY_ID, LOCATION.LOCATION_ID,
LOCATION.SHELF_NUMBER, LOCATION.FLOOR_NUMBER,
LOCATION.BUILDING_NUMBER
FROM COPY
INNER JOIN ITEM ON ITEM.ITEM_ID = COPY.ITEM_ID
INNER JOIN LOCATION ON LOCATION.LOCATION_ID = COPY.LOCATION_ID
WHERE ITEM.ITEM_ID = 8459457295;
```



	ITEM_NAME	COPY_ID	LOCATION_ID	SHELF_NUMBER	FLOOR_NUMBER	BUILDING_NUMBER
1	Exploring Geography	389	4584737598	12	2	3

This query displays all the loans that are overdue.

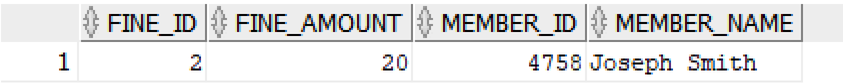
```
SELECT LOAN_ID, TIME_BORROWED + COPY_TYPE.LOAN_PERIOD AS DUE_DATE,
MEMBER.MEMBER_ID, MEMBER.MEMBER_NAME
FROM LOAN
INNER JOIN COPY ON COPY.COPY_ID = LOAN.COPY_ID
INNER JOIN COPY_TYPE ON COPY_TYPE.COPY_TYPE = COPY.COPY_TYPE
INNER JOIN MEMBER ON MEMBER.MEMBER_ID = LOAN.MEMBER_ID
WHERE TIME_BORROWED + COPY_TYPE.LOAN_PERIOD <= SYSDATE;
```



	LOAN_ID	DUE_DATE	MEMBER_ID	MEMBER_NAME
1	432	03-AUG-21	4893	Ben Johnson
2	894	04-AUG-21	4879	Rebecca Moore
3	987	23-JUN-21	4758	Joseph Smith
4	765	12-MAR-21	4758	Joseph Smith

This query displays the fines that are over the delinquent amount (10.00) specifically by member.

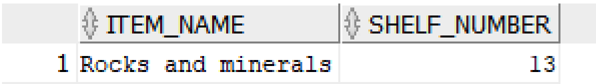
```
SELECT FINE.FINE_ID, FINE.FINE_AMOUNT, MEMBER.MEMBER_ID,
MEMBER.MEMBER_NAME
FROM FINE
INNER JOIN LOAN ON LOAN.LOAN_ID = FINE.LOAN_ID
INNER JOIN MEMBER ON MEMBER.MEMBER_ID = LOAN.MEMBER_ID
WHERE FINE.FINE_AMOUNT > 10;
```



	FINE_ID	FINE_AMOUNT	MEMBER_ID	MEMBER_NAME
1	2	20	4758	Joseph Smith

This query displays all items on a particular shelf.

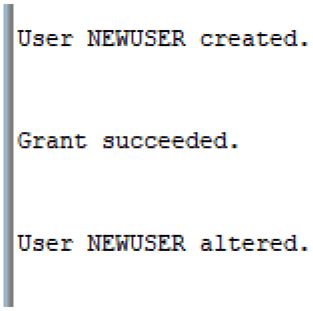
```
SELECT ITEM.ITEM_NAME, LOCATION.SHELF_NUMBER
FROM ITEM
INNER JOIN COPY ON COPY.ITEM_ID = ITEM.ITEM_ID
INNER JOIN LOCATION ON LOCATION.LOCATION_ID = COPY.LOCATION_ID
WHERE SHELF_NUMBER = 13;
```



	ITEM_NAME	SHELF_NUMBER
1	Rocks and minerals	13

This command creates a user. We decided to include this particular command because it proved integral to our completion of this coursework. We created a local Oracle database using Docker containers, which forced us to encounter a problem when creating triggers for our database: as we created all of our tables under the SYSDBA role, we were unable to create triggers for our tables. We discovered that tables created under SYS privileges cannot have triggers, which prevents the native Oracle tables from being tampered with. We had to create common users to create the triggers, which was an unexpected but important lesson.

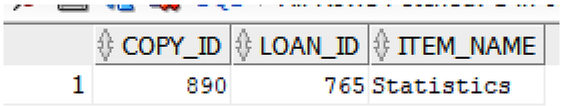
```
CREATE USER NEWUSER IDENTIFIED BY "PASSWORD";
GRANT CREATE SESSION, CREATE TABLE, CREATE TRIGGER TO "NEWUSER";
ALTER USER NEWUSER QUOTA UNLIMITED ON USERS;
```



```
User NEWUSER created.


Grant succeeded.


User NEWUSER altered.
```

This query finds the first loaned item in the library by determining the item with the earliest loan date.

```
SELECT COPY.COPY_ID, LOAN.LOAN_ID, ITEM.ITEM_NAME
FROM LOAN
INNER JOIN COPY ON COPY.COPY_ID = LOAN.COPY_ID
INNER JOIN ITEM ON ITEM.ITEM_ID = COPY.ITEM_ID
WHERE TIME_BORROWED = (SELECT MIN(TIME_BORROWED) FROM LOAN);
```

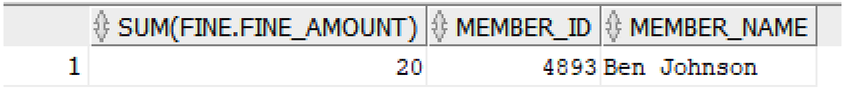


	COPY_ID	LOAN_ID	ITEM_NAME
1	890	765	Statistics

This query shows the sum of fines the member has accrued.

```
SELECT SUM(FINE.FINE_AMOUNT), MEMBER.MEMBER_ID
FROM FINE
INNER JOIN MEMBER ON MEMBER.MEMBER_ID = FINE.MEMBER_ID
GROUP BY MEMBER.MEMBER_ID
```

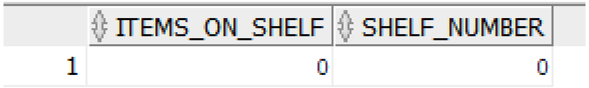
```
HAVING MEMBER_ID = 4893;
```



	SUM(FINE.FINE_AMOUNT)	MEMBER_ID	MEMBER_NAME
1	20	4893	Ben Johnson

This query shows empty shelves in the system.

```
SELECT COUNT(COPY.COPY_ID) AS ITEMS_ON_SHELF, SHELF_NUMBER
FROM LOCATION
FULL OUTER JOIN COPY ON COPY.LOCATION_ID = LOCATION.LOCATION_ID
GROUP BY COPY.COPY_ID, SHELF_NUMBER
HAVING COUNT(COPY.COPY_ID) = 0;
```

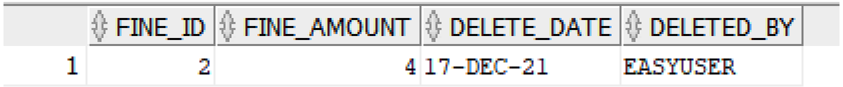


	ITEMS_ON_SHELF	SHELF_NUMBER
1	0	0

Triggers

This trigger occurs when a row in the table FINE is deleted. A record of the fine is downloaded into a fine audit table to store some of the fine's information as well as the person deleting the record.

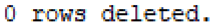
```
CREATE OR REPLACE TRIGGER FINE_AUDIT_BEFORE_DELETE
BEFORE DELETE
   ON FINE
   FOR EACH ROW
DECLARE
V_USERNAME VARCHAR(10);

BEGIN
SELECT user INTO V_USERNAME
FROM dual;

INSERT INTO FINE_AUDIT
(   FINE_ID,
   FINE_AMOUNT,
   DELETE_DATE,
   DELETED_BY)
VALUES
(
   :OLD.FINE_ID,
   :OLD.FINE_AMOUNT,
   SYSDATE,
   V_USERNAME);
END;
```



	FINE_ID	FINE_AMOUNT	DELETE_DATE	DELETED_BY
1	2	4	17-DEC-21	EASYUSER

This trigger raises an error when a user tries to delete a loan when there is a fine above 0.00 connected to that loan.

```
CREATE OR REPLACE TRIGGER RESTRICT_DELETE_LOAN_IF_FINE_PRESENT
BEFORE DELETE
   ON LOAN
   FOR EACH ROW

DECLARE
FINE_ON_LOAN NUMBER(10);

BEGIN
SELECT FINE_ID INTO FINE_ON_LOAN
FROM FINE
FULL OUTER JOIN LOAN ON LOAN.LOAN_ID = FINE.LOAN_ID
WHERE LOAN.LOAN_ID = :OLD.LOAN_ID;

IF (FINE_ON_LOAN > 0 ) then
    RAISE_APPLICATION_ERROR(-99997, 'THERE ARE OUTSTANDING FINES ON THAT
LOAN, CANNOT DELETE LOAN');
END IF;

END;
```



```
0 rows deleted.
```

This trigger creates an error when a user tries to delete a member when there is a loan present connected to that member's account.

```
CREATE OR REPLACE TRIGGER RESTRICT_DELETE_MEMBER_IF_LOAN_PRESENT
BEFORE DELETE
   ON MEMBER
   FOR EACH ROW

DECLARE
LOAN_ON_MEMBER NUMBER(10);

BEGIN
SELECT LOAN_ID INTO LOAN_ON_MEMBER
FROM LOAN
FULL OUTER JOIN MEMBER ON MEMBER.MEMBER_ID = LOAN.MEMBER_ID
WHERE LOAN.MEMBER_ID = :OLD.MEMBER_ID;

IF (LOAN_ON_MEMBER IS NOT NULL ) then
      RAISE_APPLICATION_ERROR(-99998, 'THERE ARE OUTSTANDING LOANS WITH
THAT MEMBER, CANNOT DELETE MEMBER');
END IF;
```

```
END;
```

```
0 rows deleted.
```

This trigger raises an error if the user tries to delete a location where a copy is present.

```
CREATE OR REPLACE TRIGGER RESTRICT_DELETE_LOCATION_IF_ITEM_PRESENT
BEFORE DELETE
   ON LOCATION
   FOR EACH ROW

DECLARE
COPY_ON_LOCATION NUMBER(10);

BEGIN
SELECT COPY_ID INTO COPY_ON_LOCATION
FROM COPY
FULL OUTER JOIN LOCATION ON LOCATION.LOCATION_ID = COPY.COPY_ID
WHERE COPY.COPY_ID = :OLD.LOCATION_ID;

IF (COPY_ON_LOCATION IS NOT NULL) then
        RAISE_APPLICATION_ERROR(-99999, 'COPIES ON LOCATION FOUND. MOVE
COPIES BEFORE DELETING LOCATION');
END IF;

END;
```

```
0 rows deleted.
```

Data Security

Finally, there are further database security considerations that must be discussed that pertain to this system. Database security can be defined as mechanisms that shield against intentional or accidental threats.

In this particular case, the data stored is very sensitive, as it contains information such as the full names, dates of birth and addresses of staff and students. Part of the key services offered by a database system includes ‘authorization services’, in that the database management system (DBMS) must have embedded within it a mechanism to prevent unauthorised users from accessing the database, i.e., the DBMS must ensure the database is secure. Violations of security may affect parts of the system other than the actual data, which may then in turn corrupt the database, thus database security encompasses software, hardware, software, data and people.

Threats to database security refer to any situation, either intentional or unintentional, that may negatively impact a system, and subsequently the organization. The college must therefore dedicate time to identify the most severe threats. Such threats may include a party **using another person's means of access**, for example an internal or external unauthorised party accessing the sensitive personal data of staff and students, which may result in loss of confidentiality, theft and fraud and loss of privacy for the organisation. It may also include unauthorised **program alteration,** resulting in loss of integrity of the organisation and loss of availability of the system, as the system will subsequently need to undergo thorough maintenance to fix this issue, meaning it will be unavailable to those who need it most (staff and students of the college) during this time.

The recovery that the organisation must undergo after the success of a threat depends on several factors, including any existing countermeasures for multi-user environments. In this case, the college must identify the types of threats that it is likely to encounter (e.g. those presented above) and then create strategies and countermeasures against these. Examples of computer-based countermeasures that would be potentially successful for the college include **authorization**, whereby specific privileges will be granted to specific subjects to grant them valid access to the system. The only authorised users of this library system should be the **teaching** and **IT** staff of the college as well as its students, therefore making it unlikely that any party outside of these could access the sensitive information held within the database. Moreover, our involvement of different **views** of the database also acts as a flexible countermeasure, as it conceals parts of the database from particular (group of) users. For example the list of suspended library users is only available to the staff, not to students, as they could potentially use this information for negative purposes such as to mock the suspended users. Views do not physically exist in the database, they are derived from base relations, thus views are more restrictive than offering certain privileges to users on the base relation (Connolly and Begg, 2014).

This discussion is imperative, partly as the college must comply with the UK act 2018, which controls how personal information is used by organisations, the government and businesses. This Act includes strict rules, such as that the college must ensure information is ‘handled in a way that ensures appropriate security’ and that users have the right to ‘be informed about how [their] data is being used’ (GOV.UK, n.d.). Thus, the college must establish a culture of , and policy for, legal and ethical data stewardship, including regular assessments of how changes in legislations affect the organisation, and swift and thorough treatment of lapses in legal and ethical behaviour (Connolly and Begg, 2014), for example, employee suspension if employees intentionally breach these rules.

References

Connolly, T. and Begg, C., 2014. *Database systems*. 6th ed. Boston: Pearson Education, pp.609 - 638.

GOV.UK, n.d. *Data protection*. [online] GOV.UK. Available at: <https://www.gov.uk/data-protection> [Accessed 17 December 2021].