Library Database System

Group #15

Group Members: Jack Peachey, Tzu Wang, Zhao-An Wang, Yuzo Makitani

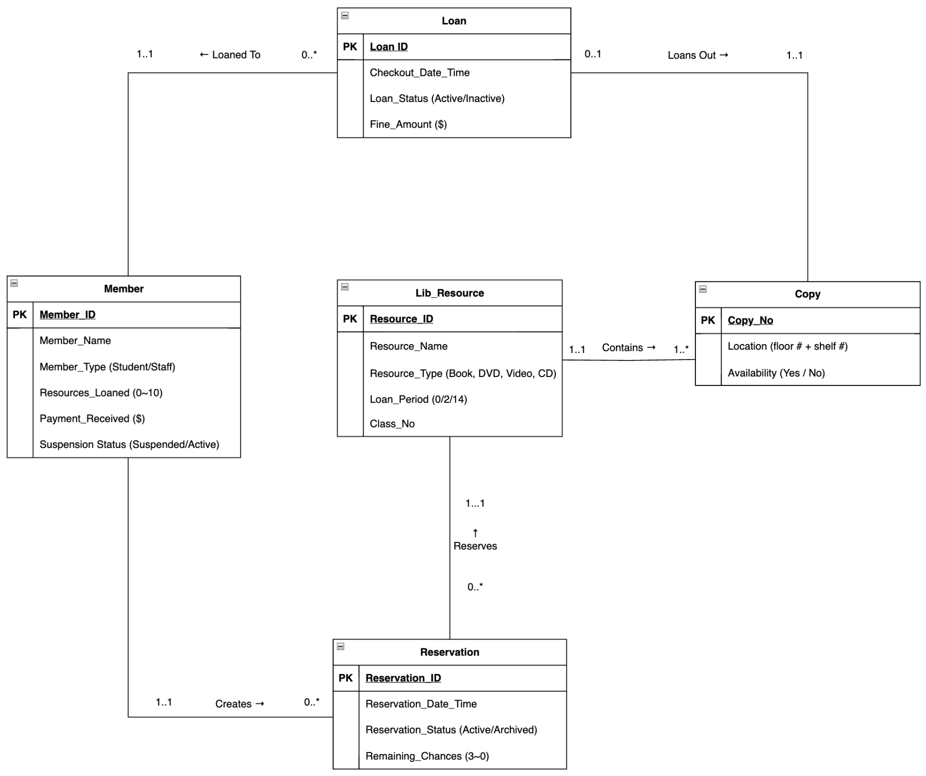
Table of Contents

Table of Contents 2
Statement on Teamwork 3
Section 1a. Conceptual Schema (ERD) 3
    Summary of Conceptual Schema 3
    Assumptions 4
    Justification of Entities and Attributes 5
Section 1b. Relational Database Schema 9
Section 1c. Normalized Design 13
Section 2a. List of all Create Table Commands 17
Section 2b. Sample Test Data 19
Section 2c. Four View Definitions 25
Section 2d. Twelve SQL Queries with Output Strings 30
Section 2e. Testing Table Constraints (Optional) 36
Bibliography 38

Statement on Teamwork

All group members of this project, Jack Peachey, Tzu Wang, Zhao-An Wang, and Yuzo Makitani, certify that we worked together equally on this coursework and that all members deserve the same grade.

Section 1a. Conceptual Schema (ERD)



Summary of Conceptual Schema

We chose five entities to represent the library database system:

- Member: Stores all members, both student and staff. It tracks the member name, member type, number of active loans, payments received, and suspension status. Members create reservations, make loans, and pay fines.
- Resource: Keeps a record of all resources in the library. It tracks the resource name, type of resource, its allowed loan period, and its class number. Resources can be a name of a book.

- Copy: A separate entity where each copy is associated with a resource. It tracks the copy number, location, and availability. Copies are the physical copy for a particular book name.
- Reservation: Stores reservations, the date and time the reservation was made, whether the reservation is active or archived, and the remaining chances a member has to pick up the resource. Reservations reserve resources and notify members of available resources.
- Loan: An entity which tracks loans and fines. It stores the checkout date and time, status of the loan, and fine amount. Loans loan out copies.

Justification of each entity and the placement of key attributes such as floor/shelf and fines is justified in the Justification of Entities section below.

Assumptions

We have made the following assumptions for our DBMS:

- A member can only request a resource and has no control over which copy of the resource is loaned to them. This is why Reservation is linked to Resource, while the Loan is linked to Copy.
- A member must pay off the entire fine amount associated with a loan at once. There is no partial payment allowed. This is why there is no separate Fine entity and instead the fine amount is tracked for each loan.
- A reservation is created by requesting a resource online or at a library kiosk.
- A loan is created when the item is physically checked out at the library counter.
- When a member checks out a copy of the resource from the library, the reservation status becomes archived. In the future if memory or query time is a concern, the reservation can be deleted instead of becoming archived.
- A reserved resource cannot be checked out by anyone other than the member who reserved the resource, if there are no more spare copies. This means that the application layer will allow members to check out copies for a particular resource as long as:

$$N_{available\ copies} > N_{active\ reservations}$$

- If a member requests a resource and it isn't currently available, a reservation is generated by the application.
- When a member checks out a resource at the counter, a request is made of the resource. If a member is not suspended, has not exceeded their quota of resources, and reservation criteria specified above are met, the system directly generates a loan, bypassing the reservation system.
- There was no requirement specified for limiting the number of reservations a member can make.

Justification of Entities and Attributes

Justification of member entity

A member entity is required to store distinct information about the member such as their name, and the member's privileges such as member type and their suspension status.

We chose to keep students and staff as a single entity because they store the same attributes. For both students and staff, we need to track their name, type, resources loaned, how much payment has been received, and their suspension status. In this system, the only difference between students and staff are that students can borrow up to 5 concurrent resources, while staff can borrow up to 10.

There is a special case where a person is both student and staff, such as a Teachers Assistant. In order to eliminate ambiguity of what permissions a student/staff member has, we have allocated three options for the attribute Member_Type: student, staff, and student_staff. We have assumed that student_staff have the same permissions as staff, but this can be changed by the customer in the application layer.

Advantage of storing member type as an attribute instead of separating student and staff entities

We store Member_Type as an attribute within the Member entity because it is a more robust and future-proofed solution than having separate entities for each member type. This allows the customer to add another member type in the future without adding another table. For example, the customer may want to separate the students into undergraduate and postgraduate member types in the future and allow them to access only certain resources in the library. A real-life example is at the Queen Mary library, where postgraduates badge into postgraduate only study rooms. Another scenario is the library adds another member type called library staff and gives them access to a longer checkout time than ordinary staff members as a worker benefit.

Justification of payment received attribute

We include a payment received attribute in members as a convenient way to track total payments received from a member. In our conversations with Chathura, it was recommended we keep a payment received attribute as we do not have fines. Note that in the physical implementation, fine payments are directly updated into the Loan_Active table, while past fines are kept in the Loan_Archive table. This means Fine from Loan_Archive can be queried and Fine from Loan_Active subtracted from it to get the same Payment_Received in Member:

$$Payment\ Received\ = Sum(Fine\ in\ Loan\ Archive)\ -\ Sum(Fine\ in\ Loan\ Active)$$

Justification of resource entity

The resource entity is required to store distinct information about the various resources available to be loaned out by the library, such as their available loan period.

In this entity, we track unique information about each resource: the type of resource it is (book, DVD, video, or CD), its available loan period (2 days, 2 weeks, or library only), and its class number.

One can make the argument that we should have another attribute called class name. This is definitely feasible, but we have excluded the class name because only class number was part of the requirement.

If we were to include class name, it would be a part of a separate entity with class number as the PK, and the class name as the dependent attribute of class number. Class name cannot be included together with class number inside the resource entity because that would break 3NF, which states there must be no transitive dependencies in the relation.

Justification of copy entity

The copy entity is required to store distinct information about each individual copy of a resource, namely its number, location, availability, the number of times it has been checked out, and the date it will become available.

Justification of separating copy and resource entities

The copy entity must exist separate of the resource entity because members must reserve resources (such as the book “Harry Potter and the Chamber of Secrets”) but need to loan individual copies (such as the copy “Harry Potter and the Chamber of Secrets Copy #4). In our discussions with Dr. Stockman, it was mentioned explicitly that members should not worry about which copy to choose when making a reservation. With copy and resource entities separated, the system makes the decision for which copy to allocate to the member.

Advantage of tracking a location inside the copy instead of resource

We track the location of a copy (its floor and shelf number) as attributes inside Copy instead of in the Resource because of instances where copies are not organized next to each other. The most prominent example which illustrates this is encyclopedia collections. A library may carry multiple copies of an encyclopedia, but it will usually group the books together by copy instead of by resource. Furthermore, encyclopedias may be so large that even one version can take up multiple shelves inside a library.

To illustrate this example, an Encyclopedia (E) may be divided into 7 books (i~vii), with 2 copies (1 or 2) each. The encyclopedia can be grouped as follows:

Shelf 18: Ei-1, Eii-1, Eiii-1, Eiv-1, Ev-1, Evi-1, Evii-1

Shelf 19: Ei-2, Eii-2, Eiii-2, Eiv-2, Ev-2, Evi-2, Evii-2

In this example, Evii-1 and Evii-2 are the same resource, but each copy is located on a different shelf. This means tracking location by copy is a more truthful representation of the physical location than tracking a resource which may be spread out over multiple locations.

Justification of loan entity

The loan entity is required to store distinct information about who borrowed which copy of a resource, when the resource must be returned, and how much fine is owed for a particular loan. The loan entity allows a member to borrow a particular copy of a resource while the reservation entity allows a member to make a reservation of a resource without needing to specify the copy they want to reserve.

Advantage of tracking fine amount within loan entity

The fine amount is tracked directly inside the loan entity. This treats all loans as a fine of default value zero. The advantage of this system is that it eliminates the need for a separate fine table, which can take up extra memory because it will need to store foreign keys Member_ID and Loan_ID. However, if few members have fines, creating a separate Fine entity can be advantageous to speed up the query process of fines and reduce the extra memory required by storing a fine value for all loans.

The fine attribute is not a derived attribute because it is dependent on two factors: the fine incurred by the member for a particular loan, and the amount paid off by the member (in our case, must be exactly the fine amount):

$$Fine\ =\ \$1.00 \cdot Days\ Overdue - Amount\ Paid\ by\ Member$$

Where:

$$Days\ Overdue\ =\ (Current\ Date\ Time\ -\ Checkout\ Date\ Time)\ -\ Loan\ Period$$

The amount paid by member is an input the DBMS receives from the application layer, therefore not derivable. However, the days overdue and therefore the fine incurred by a member can be derived by the above formula.

Loan Archive vs Loan Active

There is no differentiation between Loan Archive and Loan Active in the ERD, because Loan Archive is not a necessary table. Loan Archive is implemented in the physical level (part 2 of coursework) to optimize query speed on active loans.

A loan is active if it has not been returned or a fine has not been paid off yet. When a fine is paid off and the copy is returned, the loan is removed from the loan active table. However, the loan archive table will maintain the fine – this enables us to keep a history of past loans and fine amounts (like Order History on Amazon).

Justification of reservation entity

The reservation entity is required to store distinct information about when a reservation was made, its status, and how many times a member has left remaining to check out a resource. Unlike the loan entity, it connects a member to a resource instead of an individual copy of a resource. The reservation entity allows the system to allocate resources to members and notify members when resources become available.

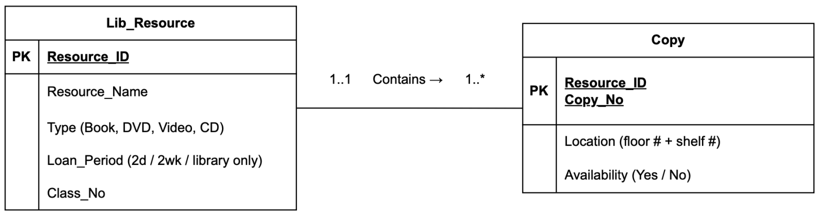
Reservation Status

A reservation is active by default until all remaining chances are exhausted or the member cancels their reservation. Inactive reservations are archived for now, but this can be changed to delete inactive reservations in the future if the customer does not have a need to store them. If they need to be stored but become large, we can create a reservation archive table in the future.

Section 1b. Relational Database Schema

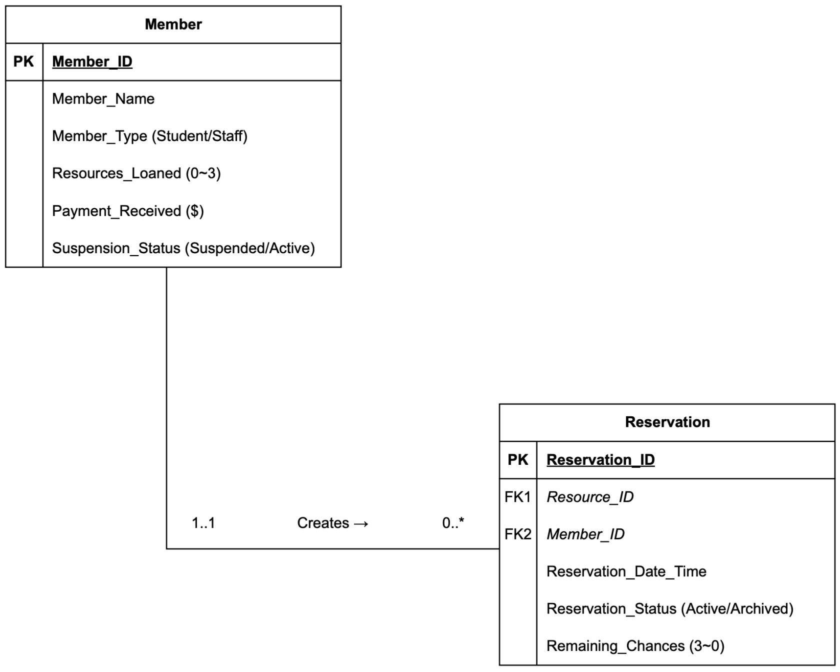
Step 1: One to Many Relation

- Moved the PK of Resource (Resource_ID) into Copy as a FK
- Since Copy is a weak entity, Resource_ID is used as part of the composite PK



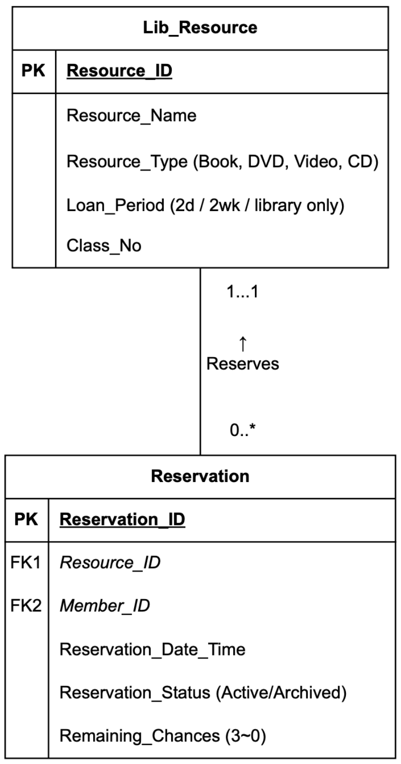
Step 2: One to Many Relation

- Moved the PK of Member (Member_ID) into Reservation as a FK



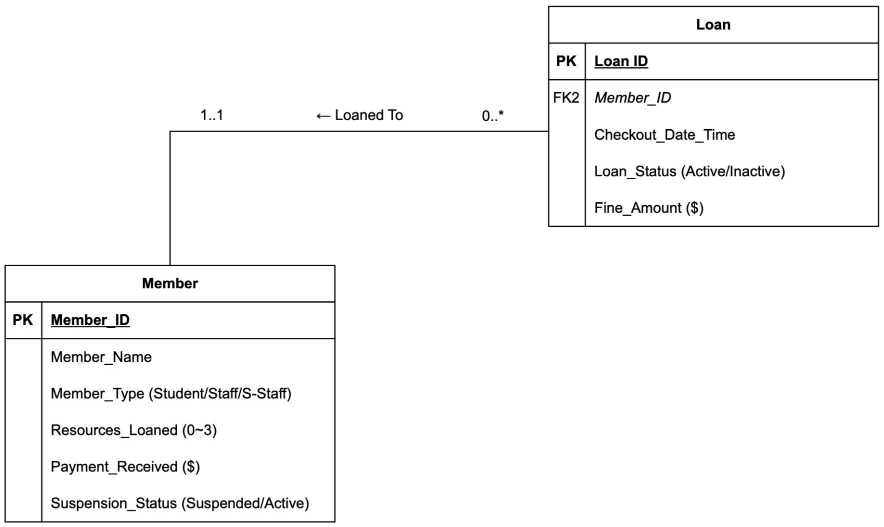
Step 3: One to Many Relation

- Moved the PK of Resource (Resource_ID) into Reservation as a FK


Lib_Resource
PK
Resource_ID
Resource_Name
Resource_Type (Book, DVD, Video, CD)
Loan_Period (2d / 2wk / library only)
Class_No
1...1
↑
Reserves
0..*
Reservation
PK
Reservation_ID
FK1
Resource_ID
FK2
Member_ID
Reservation_Date_Time
Reservation_Status (Active/Archived)
Remaining_Chances (3~0)

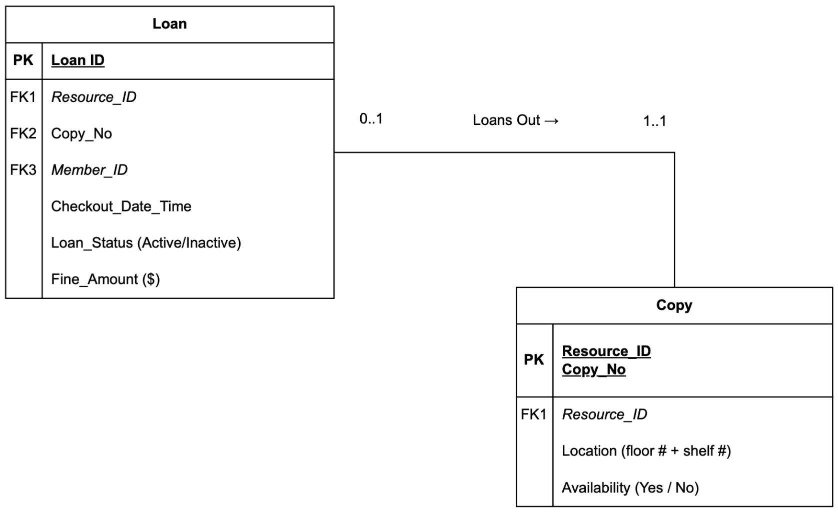
Step 4: One to Many Relation

- Moved the PK of Member (Member_ID) into Loan as a FK

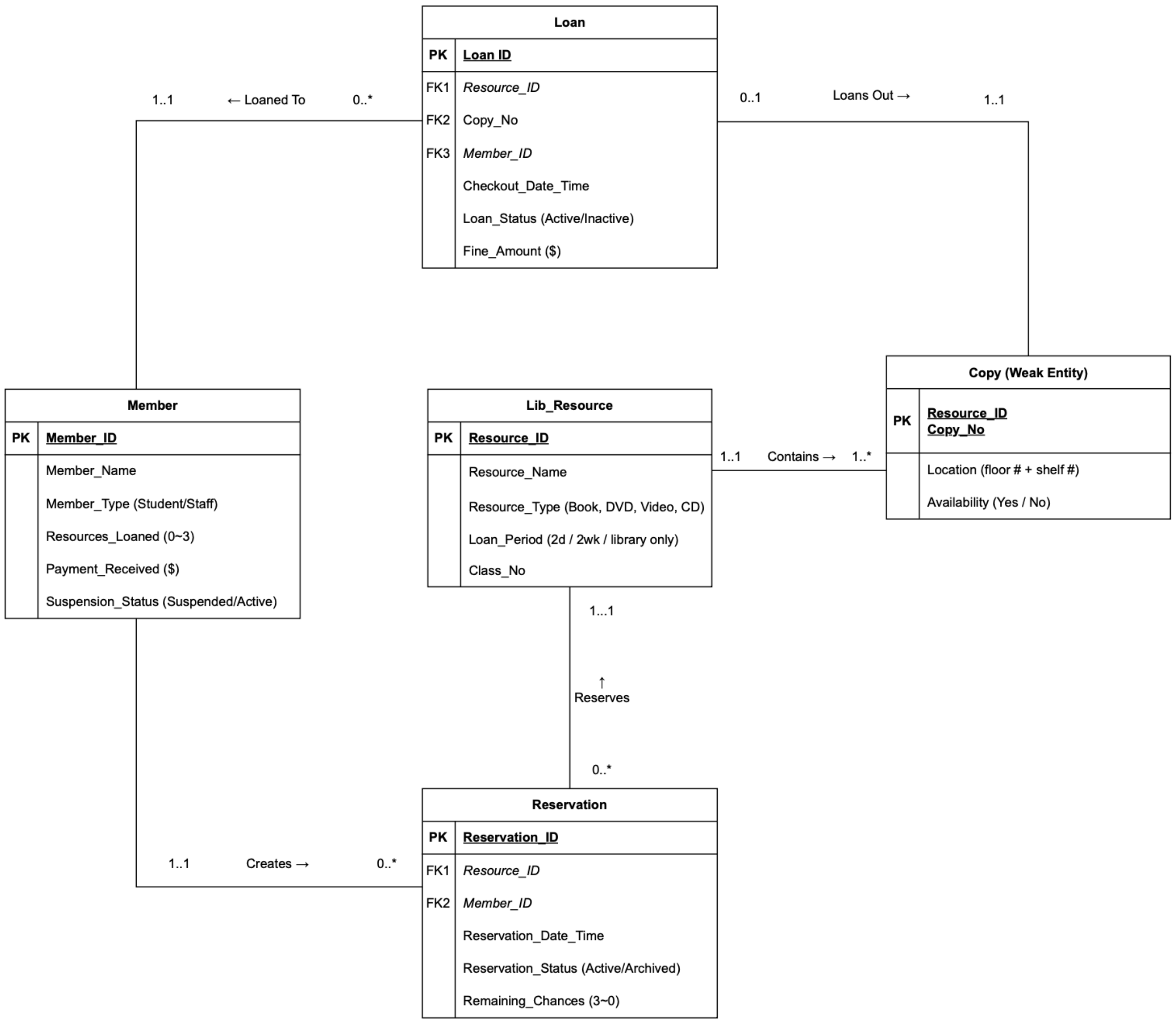


Step 5: One to One Relation

- Moved the PKs of Copy (Resource_ID, Copy_No) into Loan as FKs



Relational Schema After Mapping



MEMBER(<u>Member_ID</u> {PK}, Member_Name, Member_Type, Resources_Loaned, Payment_Received)
RESERVATION(<u>Reservation ID</u> {PK}, Resource_ID {FK}, Member_ID {FK}, Reservation_Date_Time, Reservation_Status, Remaining_Chances)
LIB_RESOURCE(<u>Resource_ID</u> {PK}, Resource_Name, Resource_Type, Loan_Period, Class_No)
COPY(<u>Resource_ID_Copy_No</u> {PK}, Resource_ID {FK}, Copy_No, Location, Availability, Checkout_Counter)
LOAN(<u>Loan_ID</u> {PK}, Resource_ID_Copy_No {FK}, Member_ID {FK}, Checkout_Date_Time, Loan_Status, Fine_Amount)

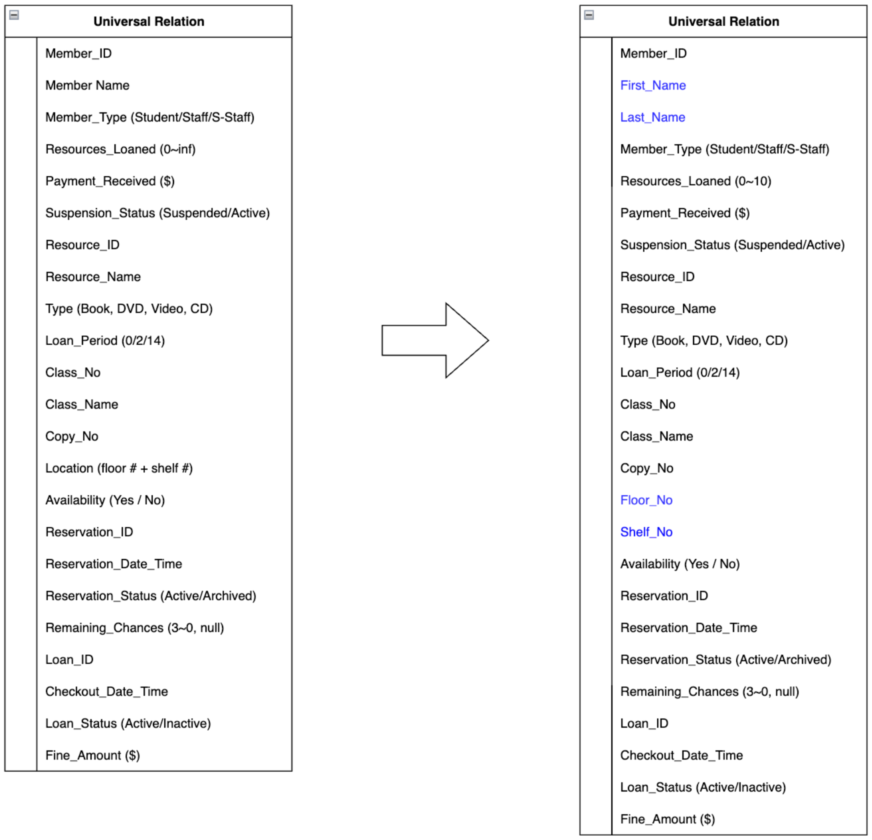
Section 1c. Normalized Design

1NF - Identify and remove repeating groups (multi-valued attributes)

Divided the following attributes into their atomic attributes:

- Member Name into Member_First_Name and Member_Last_Name
- Location into Floor_No and Shelf_No
- Note that Resource_ID, Resource_ID_Copy_No, Reservation_ID, and Loan_ID are tracked as candidate keys

Blue font indicates updated values after normalization. Diagram is used for clarity, with relational schema at bottom of the page.



Universal Relation
Member_ID
Member Name
Member_Type (Student/Staff/S-Staff)
Resources_Loaned (0~inf)
Payment_Received ($)
Suspension_Status (Suspended/Active)
Resource_ID
Resource_Name
Type (Book, DVD, Video, CD)
Loan_Period (0/2/14)
Class_No
Class_Name
Copy_No
Location (floor # + shelf #)
Availability (Yes / No)
Reservation_ID
Reservation_Date_Time
Reservation_Status (Active/Archived)
Remaining_Chances (3~0, null)
Loan_ID
Checkout_Date_Time
Loan_Status (Active/Inactive)
Fine_Amount ($)

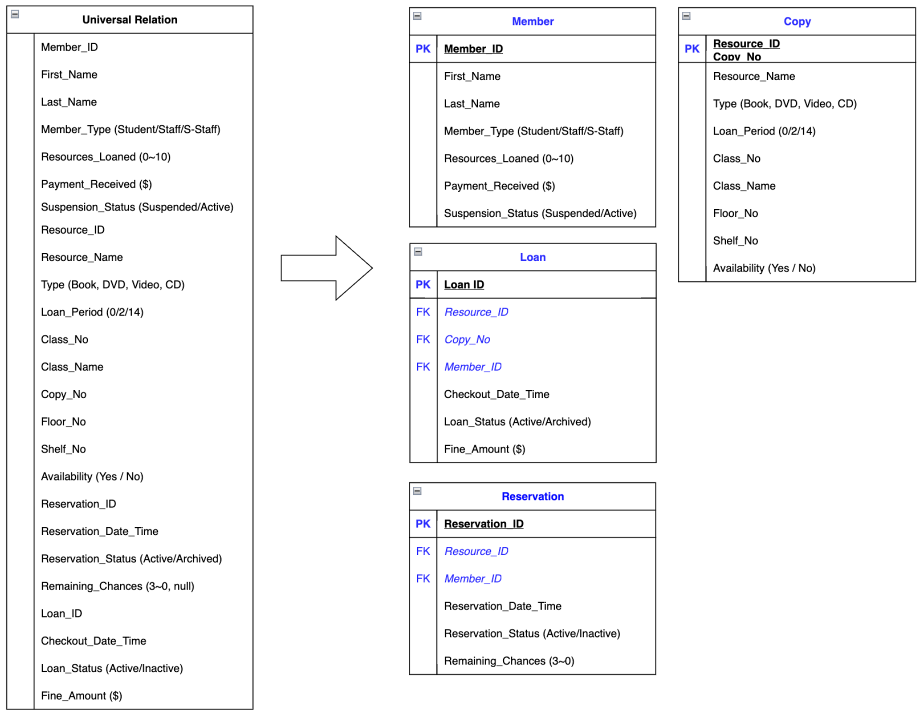
Universal Relation
Member_ID
First_Name
Last_Name
Member_Type (Student/Staff/S-Staff)
Resources_Loaned (0~10)
Payment_Received ($)
Suspension_Status (Suspended/Active)
Resource_ID
Resource_Name
Type (Book, DVD, Video, CD)
Loan_Period (0/2/14)
Class_No
Class_Name
Copy_No
Floor_No
Shelf_No
Availability (Yes / No)
Reservation_ID
Reservation_Date_Time
Reservation_Status (Active/Archived)
Remaining_Chances (3~0, null)
Loan_ID
Checkout_Date_Time
Loan_Status (Active/Inactive)
Fine_Amount ($)

UNIVERSAL RELATION (Member_ID, First_Name, Last_Name, Member_Type, Resources_Loaned, Payment_Received, Suspension_Status,
Reservation ID, Member_ID {FK}, Reservation_Date_Time, Reservation_Status, Remaining_Chances,
Resource_ID, Resource_Name, Resource_Type, Loan_Period, Class_No, Class_Name,
Resource_ID, Copy_No, Floor_No, Shelf_No, Availability,
Loan_ID, Checkout_Date_Time, Loan_Status, Fine_Amount)

2NF - Every other attribute must be functionally dependent on one primary key

Established four relations which ensure each attribute is functionally dependent on only one primary key

- From Member_ID, we can determine the member's first name, last name, membership type, total payment received, and suspension status
- From Reservation_ID, we can determine the reservation date and time, reservation status, and how many remaining chances the member has left to pick up the book
- From Loan_ID, we can determine the checkout date and time, loan status, and the fine amount
- From Resource_ID, Copy_ID, we can determine the resource name, type, maximum loan period, class number and name, its floor and shelf, and availability



MEMBER (Member_ID {PK}, First_Name, Last_Name, Member_Type, Resources_Loaned, Payment_Received, Suspension_Status)
RESERVATION (Reservation ID {PK}, Resource_ID {FK}, Member_ID {FK}, Reservation_Date_Time, Reservation_Status, Remaining_Chances)
COPY (Resource_ID, Copy_No {PK}, Copy_No, Location, Availability, Resource_Name, Resource_Type, Loan_Period, Class_No)
LOAN (Loan_ID {PK}, Resource_ID {FK}, Copy_No {FK}, Member_ID {FK}, Checkout_Date_Time, Loan_Status, Fine_Amount)

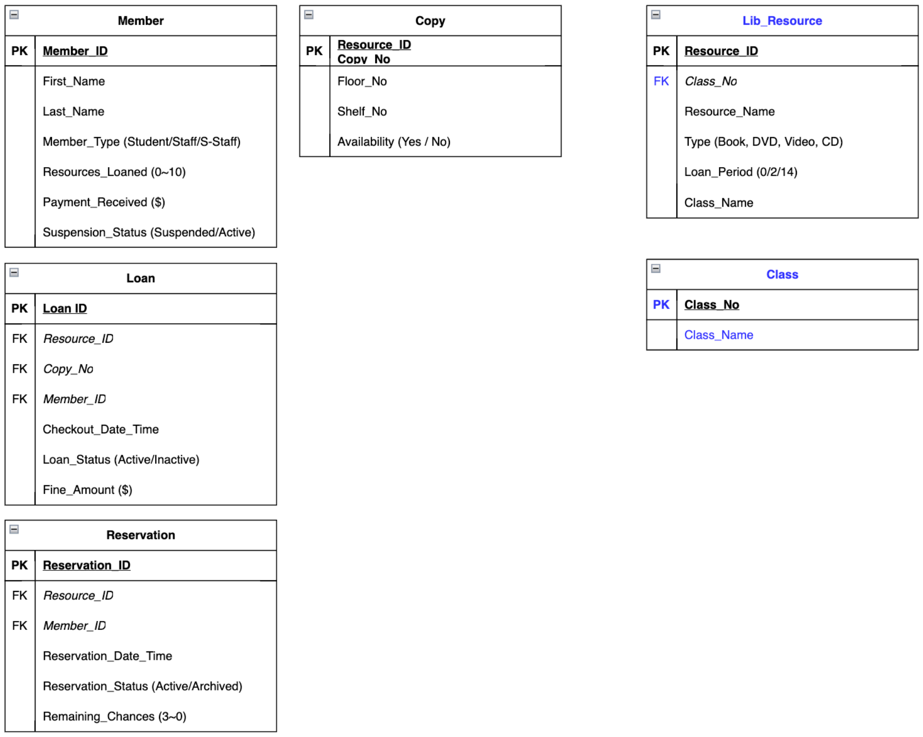
3NF - Eliminate transitive dependencies

Separated the transitive dependencies Resource_Name, Type, Loan_Period, Class_No, and Class_Name, which are directly dependent on Resource_ID but only transitively dependent on Resource_ID, Copy_ID

- We do not need information about the copy number to determine the name, type, maximum loand period, and class details about a resource

Separated the transitive dependency Class_Name which is directly dependent on Class_No and only transitively dependent on Resource_ID

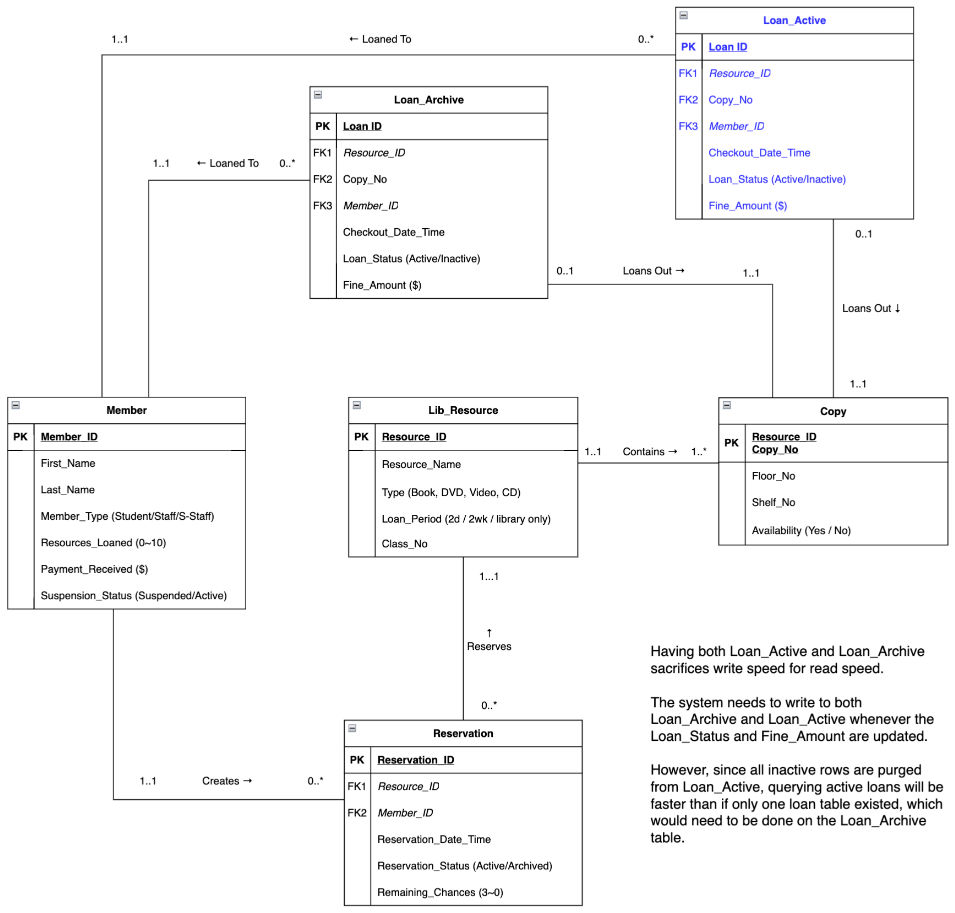
- Note that Class_Name is not a part of our relational schema because it is not a part of the coursework requirements to track Class_Name - shown here to illustrate 3NF



MEMBER (Member_ID {PK}, First_Name, Last_Name, Member_Type, Resources_Loaned, Payment_Received, Suspension_Status)
RESERVATION (Reservation ID {PK}, Resource_ID {FK}, Member_ID {FK}, Reservation_Date_Time, Reservation_Status, Remaining_Chances)
COPY (Resource_ID, Copy_No {PK}, Resource_ID {FK}, Copy_No, Location, Availability)
LOAN (Loan_ID {PK}, Resource_ID_Copy_No {FK}, Member_ID {FK}, Checkout_Date_Time, Loan_Status, Fine_Amount)
LIB_RESOURCE (Resource_ID {PK}, Resource_Name, Type, Loan_Period, Class_No)
CLASS (Class_No {PK}, Class_Name)

Final relational schema after relational mapping and normalization

- Removed the Class relation because class name is not specified as a requirement in the coursework
- Loan_Active schema is added for the purpose of efficiency in the physical system. Loan_Active will hold only active loans while Loan_Archive stores all loans, both active and inactive. This enables current loan queries to be executed quickly on Loan_Active



Having both Loan_Active and Loan_Archive sacrifices write speed for read speed.

The system needs to write to both Loan_Archive and Loan_Active whenever the Loan_Status and Fine_Amount are updated.

However, since all inactive rows are purged from Loan_Active, querying active loans will be faster than if only one loan table existed, which would need to be done on the Loan_Archive table.

MEMBER (Member_ID {PK}, First_Name, Last_Name, Member_Type, Resources_Loaned, Payment_Received, Suspension_Status)
RESERVATION (Reservation ID {PK}, Resource_ID {FK}, Member_ID {FK}, Reservation_Date_Time, Reservation_Status, Remaining_Chances)
LIB_RESOURCE (Resource_ID {PK}, Resource_Name, Type, Loan_Period, Class_No)
COPY (Resource_ID, Copy_No {PK}, Resource_ID {FK}, Copy_No, Location, Availability)
LOAN_ARCHIVE (Loan_ID {PK}, Resource_ID_Copy_No {FK}, Member_ID {FK}, Checkout_Date_Time, Loan_Status, Fine_Amount)
LOAN_ACTIVE (Loan_ID {PK}, Resource_ID_Copy_No {FK}, Member_ID {FK}, Checkout_Date_Time, Loan_Status, Fine_Amount)

Section 2a. List of all Create Table Commands

```
CREATE TABLE MEMBER(
   MEMBER_ID VARCHAR2(10) PRIMARY KEY,
   MEMBER_TYPE VARCHAR2(15) CONSTRAINT CHECK_MEMBER_TYPE CHECK
(MEMBER_TYPE IN ('student', 'staff', 'student_staff')),
   RESOURCES_LOANED NUMBER(2, 0),
   PAYMENT_RECEIVED NUMBER(4, 0),
   SUSPENSION_STATUS NUMBER(1, 0) CONSTRAINT CHECK_SUSPENSION_STATUS
CHECK (SUSPENSION_STATUS BETWEEN 0 AND 1),
   FIRST_NAME VARCHAR2(20),
   LAST_NAME VARCHAR2(20),
   CONSTRAINT CHECK_RESOURCES_LOANED CHECK((MEMBER_TYPE = 'student' AND
RESOURCES_LOANED <= 5) OR
   ((MEMBER_TYPE = 'staff' OR  MEMBER_TYPE = 'student_staff') AND
RESOURCES_LOANED <= 10))
);

CREATE TABLE LIB_RESOURCE(
   RESOURCE_ID VARCHAR2(10) PRIMARY KEY,
   RESOURCE_NAME VARCHAR2(50),
   RESOURCE_TYPE VARCHAR2(10),
   LOAN_PERIOD NUMBER(2),
   CLASS_NO NUMBER(5, 0),
   CONSTRAINT CHECK_LOAN_PERIOD CHECK (LOAN_PERIOD >= 0 AND
LOAN_PERIOD <= 14)
);

CREATE TABLE COPY(
   COPY_NO NUMBER(3, 0),
   RESOURCE_ID VARCHAR2(10),
   FLOOR NUMBER(2, 0),
   SHELF NUMBER(3, 0),
   AVAILABILITY NUMBER(1, 0) CONSTRAINT CHECK_AVAILABILITY CHECK
(AVAILABILITY BETWEEN 0 AND 1),
   PRIMARY KEY(COPY_NO, RESOURCE_ID),
   CONSTRAINT FK_RESOURCE FOREIGN KEY (RESOURCE_ID) REFERENCES
LIB_RESOURCE(RESOURCE_ID)
);

CREATE TABLE LOAN_ACTIVE(
   LOAN_ID VARCHAR2(10) PRIMARY KEY,
   MEMBER_ID VARCHAR2(10),
   CHECKOUT_DATE_TIME TIMESTAMP,
   LOAN_STATUS NUMBER(1, 0) CONSTRAINT CHECK_LOAN_STATUS CHECK
(LOAN_STATUS BETWEEN 0 AND 1),
   FINE_AMOUNT NUMBER(4, 0),
   COPY_NO NUMBER(3, 0),
   RESOURCE_ID VARCHAR2(10),
   CONSTRAINT FK_MEMBER_ID FOREIGN KEY (MEMBER_ID) REFERENCES
MEMBER(MEMBER_ID),
   CONSTRAINT FK_RESOURCE_ID_COPY_NO FOREIGN KEY (COPY_NO,
RESOURCE_ID) REFERENCES COPY(COPY_NO, RESOURCE_ID)
);

CREATE TABLE LOAN_ARCHIVE(
   LOAN_ID VARCHAR2(10) PRIMARY KEY,
   MEMBER_ID VARCHAR2(10),
   CHECKOUT_DATE_TIME TIMESTAMP,
   LOAN_STATUS NUMBER(1, 0) CONSTRAINT CHECK_LOAN_STATUS_2 CHECK
(LOAN_STATUS BETWEEN 0 AND 1),
   FINE_AMOUNT NUMBER(4, 0),
   COPY_NO NUMBER(3, 0),
   RESOURCE_ID VARCHAR2(10),
   CONSTRAINT FK_MEMBER_ID_2 FOREIGN KEY (MEMBER_ID) REFERENCES
MEMBER(MEMBER_ID),
   CONSTRAINT FK_RESOURCE_ID_COPY_NO_2 FOREIGN KEY (COPY_NO,
RESOURCE_ID) REFERENCES COPY(COPY_NO, RESOURCE_ID)
);

CREATE TABLE RESERVATION(
   RESERVATION_ID VARCHAR2(10) PRIMARY KEY,
   RESERVATION_DATE_TIME TIMESTAMP,
   RESERVATION_STATUS NUMBER(1, 0) CONSTRAINT CHECK_RESERVATION_STATUS
CHECK (RESERVATION_STATUS BETWEEN 0 AND 1),
   REMAINING_CHANCES NUMBER(1, 0) CONSTRAINT CHECK_REMAINING_CHANCES
CHECK (REMAINING_CHANCES >= 0 AND REMAINING_CHANCES <= 3),
   MEMBER_ID VARCHAR2(10),
   RESOURCE_ID VARCHAR2(10),
   CONSTRAINT FK_RESOURCE_ID_RES FOREIGN KEY (RESOURCE_ID) REFERENCES
LIB_RESOURCE(RESOURCE_ID),
   CONSTRAINT FK_MEMBER_ID_RES FOREIGN KEY (MEMBER_ID) REFERENCES
MEMBER(MEMBER_ID)
);
```

Section 2b. Sample Test Data

Table	Attribute	Permitted Values
Member	Member_ID	Unique ID
	First_Name	String
	Last_Name	String
	Member_Type	Staff, student, or student_staff
	Resources_Loaned	0 ~ 5 (students), 0 ~ 10 (staff and student_staff)
	Payment_Received	0~9999 ($)
	Suspension Status	0 (active), 1 (suspended)
Lib_Resource	Resource_ID	Unique ID
	Resource_Name	String
	Type	Book, DVD, Video, or CD
	Loan_Period	0 (library only), 2, or 14 days
	Class_No	Number
Copy	Resource ID	Unique ID
	Copy_No	0~999
	Floor NO.	0~99
	shelf NO.	0~999
	Availability	0 (unavailable), 1 (available)
Loan_Archive	Loan ID	Unique ID
	Member ID	Unique ID
	Check Out Datetime	yyyy/mm/dd hh:mi:ss
	Loan Status	0 (inactive), 1 (active)
	Fine Amount	0~9999 ($)
	Copy NO.	0~999
	Resource ID	Unique ID
Loan_Active	Loan ID	Unique ID
	Member ID	Unique ID
	Check Out Datetime	yyyy/mm/dd hh:mi:ss
	Loan Status	0 (inactive), 1 (active)
	Fine Amount	0~9999 ($)
	Copy NO.	0~999
	Resource ID	Unique ID
Reservation	Reservation ID	Unique ID
	Reservation Date Time	yyyy/mm/dd hh:mi:ss
	Reservation Status	0 (cancelled), 1 (active)
	Remaining Chances	0 (canceled) ~ 3 (default value)
	Member ID	Unique ID
	Resource ID	Unique ID

```
INSERT ALL
    INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED,
PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME)
VALUES('A123456789', 'staff', 0, 3, 0, 'Yuzo', 'Makitani')
    INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED,
PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME)
VALUES('B234567890', 'staff', 2, 0, 1, 'Jeff', 'Wang')
   INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED,
PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME)
VALUES('C345678901', 'staff', 2, 3, 0, 'Rick', 'M')
   INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED,
PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME)
VALUES('D456789012', 'student_staff', 3, 5, 1, 'Jack', 'Peachy')
    INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED,
PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME)
VALUES('E567890123', 'student_staff', 1, 10, 0, 'Linda', 'L')
   INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED,
PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME)
VALUES('F678901234', 'student_staff', 2, 0, 0, 'Denise', 'H')
    INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED,
PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME)
VALUES('G789012345', 'student', 0, 0, 0, 'Joel', 'Wang')
    INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED,
PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME)
VALUES('H890123456', 'student', 2, 0, 0, 'Lia', 'W')
    INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED,
PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME)
VALUES('I901234567', 'student', 1, 0, 0, 'Wallis', 'L')
   INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED,
PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME)
VALUES('J012345678', 'student', 0, 0, 0, 'Steve', 'J')
SELECT * FROM dual;


INSERT ALL
    INTO LIB_RESOURCE (RESOURCE_ID, RESOURCE_NAME , RESOURCE_TYPE,
LOAN_PERIOD, CLASS_NO) VALUES('123456789A', 'AAA', 'Book', 14, 0)
    INTO LIB_RESOURCE (RESOURCE_ID, RESOURCE_NAME , RESOURCE_TYPE,
LOAN_PERIOD, CLASS_NO) VALUES('234567890B', 'BBB', 'Book', 0, 1)
    INTO LIB_RESOURCE (RESOURCE_ID, RESOURCE_NAME , RESOURCE_TYPE,
LOAN_PERIOD, CLASS_NO) VALUES('345678901C', 'CCC', 'Video', 14, 1)
    INTO LIB_RESOURCE (RESOURCE_ID, RESOURCE_NAME , RESOURCE_TYPE,
LOAN_PERIOD, CLASS_NO) VALUES('456789012D', 'DDD', 'Video', 2, 50)
    INTO LIB_RESOURCE (RESOURCE_ID, RESOURCE_NAME , RESOURCE_TYPE,
LOAN_PERIOD, CLASS_NO) VALUES('567890123E', 'EEE', 'DVD', 2, 32)
    INTO LIB_RESOURCE (RESOURCE_ID, RESOURCE_NAME , RESOURCE_TYPE,
LOAN_PERIOD, CLASS_NO) VALUES('678901234F', 'FFF', 'DVD', 0, 32)
```

```
   INTO LIB_RESOURCE (RESOURCE_ID, RESOURCE_NAME , RESOURCE_TYPE,
LOAN_PERIOD, CLASS_NO) VALUES('789012345G', 'GGG', 'CD', 14, 18)
   INTO LIB_RESOURCE (RESOURCE_ID, RESOURCE_NAME , RESOURCE_TYPE,
LOAN_PERIOD, CLASS_NO) VALUES('890123456H', 'HHH', 'CD', 14, 0)
   INTO LIB_RESOURCE (RESOURCE_ID, RESOURCE_NAME , RESOURCE_TYPE,
LOAN_PERIOD, CLASS_NO) VALUES('000000000Z', 'ZZZ', 'Book', 2, 50)
SELECT * FROM dual;


INSERT ALL
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (0,
'123456789A', 0, 0, 0)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (1,
'123456789A', 0, 0, 0)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (2,
'123456789A', 0, 1, 0)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (0,
'234567890B', 1, 2, 1)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (1,
'234567890B', 1, 2, 1)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (0,
'345678901C', 1, 3, 0)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (0,
'456789012D', 1, 4, 0)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (1,
'456789012D', 1, 4, 0)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (2,
'456789012D', 1, 4, 1)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (3,
'456789012D', 1, 5, 1)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (0,
'567890123E', 2, 3, 1)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (1,
'567890123E', 2, 3, 0)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (0,
'678901234F', 2, 3, 1)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (0,
'789012345G', 2, 3, 0)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (1,
'789012345G', 2, 4, 0)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (2,
'789012345G', 2, 4, 1)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (0,
'890123456H', 3, 0, 0)
   INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (1,
'890123456H', 3, 0, 0)
```

```
    INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (2,
'890123456H', 3, 1, 1)
    INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (3,
'890123456H', 3, 1, 0)
    INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (4,
'890123456H', 3, 1, 0)
    INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (0,
'000000000Z', 4, 0, 1)
SELECT * FROM dual;


INSERT ALL
  INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L0', 'B234567890',
TIMESTAMP '2023-10-01 09:00:00', 1, 45, 0, '890123456H')
   INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L1', 'D456789012',
TIMESTAMP '2023-10-01 10:00:00', 1, 45, 1, '890123456H')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L2', 'A123456789',
TIMESTAMP '2023-11-01 09:00:00', 0, 3, 0, '123456789A')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L3', 'B234567890',
TIMESTAMP '2023-11-02 10:00:00', 0, 0, 0, '345678901C')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L4', 'C345678901',
TIMESTAMP '2023-11-03 11:00:00', 0, 2, 0, '456789012D')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L5', 'D456789012',
TIMESTAMP '2023-11-04 12:00:00', 0, 2, 0, '567890123E')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L6', 'E567890123',
TIMESTAMP '2023-11-05 13:00:00', 0, 0, 0, '789012345G')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L7', 'B234567890',
TIMESTAMP '2023-11-06 09:00:00', 1, 10, 2, '123456789A')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L8', 'C345678901',
TIMESTAMP '2023-11-19 08:00:08', 1, 0, 0, '345678901C')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L9', 'C345678901',
TIMESTAMP '2023-11-19 08:00:08', 1, 9, 0, '456789012D')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L10', 'D456789012',
TIMESTAMP '2023-11-20 13:19:20', 1, 8, 1, '456789012D')
```

```
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L11', 'D456789012',
TIMESTAMP '2023-11-21 09:00:00', 1, 7, 1, '567890123E')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L12', 'E567890123',
TIMESTAMP '2023-11-22 11:11:11', 1, 10, 0, '789012345G')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L13', 'F678901234',
TIMESTAMP '2023-11-23 19:00:19', 1, 0, 1, '789012345G')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L14', 'F678901234',
TIMESTAMP '2023-11-23 20:00:00', 1, 0, 3, '890123456H')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L15', 'H890123456',
TIMESTAMP '2023-11-24 02:02:02', 1, 0, 4, '890123456H')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L16', 'H890123456',
TIMESTAMP '2023-11-25 07:07:07', 1, 0, 0, '123456789A')
    INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME,
LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L17', 'I901234567',
TIMESTAMP '2023-11-26 07:07:57', 1, 0, 1, '123456789A')
SELECT * FROM dual;


INSERT ALL
   INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L0', 'B234567890', TIMESTAMP
'2023-10-01 09:00:00', 1, 45, 0, '890123456H')
   INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L1', 'D456789012', TIMESTAMP
'2023-10-01 10:00:00', 1, 45, 1, '890123456H')
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L7', 'B234567890', TIMESTAMP
'2023-11-06 09:00:00', 1, 10, 2, '123456789A')
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L8', 'C345678901', TIMESTAMP
'2023-11-19 08:00:08', 1, 0, 0, '345678901C')
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L9', 'C345678901', TIMESTAMP
'2023-11-19 08:00:08', 1, 8, 0, '456789012D')
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L10', 'D456789012', TIMESTAMP
'2023-11-20 13:19:20', 1, 5, 1, '456789012D')
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L11', 'D456789012', TIMESTAMP
'2023-11-21 09:00:00', 1, 7, 1, '567890123E')
```

```
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L12', 'E567890123', TIMESTAMP
'2023-11-22 11:11:11', 1, 0, 0, '789012345G')
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L13', 'F678901234', TIMESTAMP
'2023-11-23 19:00:19', 1, 0, 1, '789012345G')
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L14', 'F678901234', TIMESTAMP
'2023-11-23 20:00:00', 1, 0, 3, '890123456H')
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L15', 'H890123456', TIMESTAMP
'2023-11-24 02:02:02', 1, 0, 4, '890123456H')
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L16', 'H890123456', TIMESTAMP
'2023-11-25 07:07:07', 1, 0, 0, '123456789A')
    INTO LOAN_ACTIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS,
FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES ('L17', 'I901234567', TIMESTAMP
'2023-11-26 07:07:57', 1, 0, 1, '123456789A')
SELECT * FROM dual;


INSERT ALL
    INTO RESERVATION (RESERVATION_ID, RESERVATION_DATE_TIME,
RESERVATION_STATUS, REMAINING_CHANCES, MEMBER_ID, RESOURCE_ID) VALUES
('R1', TIMESTAMP '2023-11-20 08:00:00', 0, 3, 'F678901234', '345678901C')
    INTO RESERVATION (RESERVATION_ID, RESERVATION_DATE_TIME,
RESERVATION_STATUS, REMAINING_CHANCES, MEMBER_ID, RESOURCE_ID) VALUES
('R2', TIMESTAMP '2023-11-21 09:00:00', 1, 3, 'G789012345', '345678901C')
    INTO RESERVATION (RESERVATION_ID, RESERVATION_DATE_TIME,
RESERVATION_STATUS, REMAINING_CHANCES, MEMBER_ID, RESOURCE_ID) VALUES
('R3', TIMESTAMP '2023-11-22 10:00:00', 0, 0, 'A123456789', '789012345G')
    INTO RESERVATION (RESERVATION_ID, RESERVATION_DATE_TIME,
RESERVATION_STATUS, REMAINING_CHANCES, MEMBER_ID, RESOURCE_ID) VALUES
('R4', TIMESTAMP '2023-11-26 13:00:00', 1, 3, 'E567890123', '123456789A')
    INTO RESERVATION (RESERVATION_ID, RESERVATION_DATE_TIME,
RESERVATION_STATUS, REMAINING_CHANCES, MEMBER_ID, RESOURCE_ID) VALUES
('R5', TIMESTAMP '2023-11-27 14:00:00', 1, 2, 'A123456789', '890123456H')
SELECT * FROM dual;
```

Section 2c. Four View Definitions

All views were tested on November 29th.

These four view definitions are designed to help library staff users of the database system in the following key areas: unpaid fines, fines paid, lost resources and resource analytics.

Unpaid Fines View

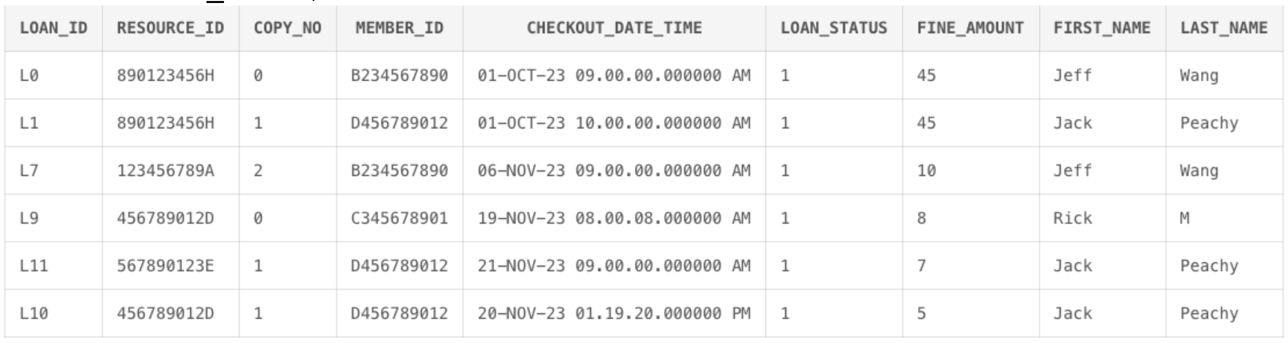
The ‘Unpaid Fines’ view outputs all fines from overdue books; all active loans for which Fine_Amount attribute greater than or equal to one.

Staff responsible for library finances will benefit from this view. Firstly, the view exhaustively shows all the money that the library is expecting to receive from outstanding fines; a feature which is useful for financial planning. Secondly, the view links each fine to a member’s name. Staff can use this name (along with the unique Member_ID) to notify library members with fines. (The database system does not store emails directly as this is beyond the coursework specification. It is assumed that member emails can be accessed accurately outside of the system when a member’s name and/or Member_ID is available.) Library staff will need to, a) let members know they have been served a fine because of a specific overdue loan and, b) provide them with the details on how to pay the fine. The system can be updated by a staff user to represent a fine having been paid off by a) adding the paid off amount to the member’s Payment_Received , and b) changing the Loan_Status of the active loan to inactive (in turn causing it to be deleted from active loan. Thus, the paid-off loan will now no longer appear in the unpaid fines view).

```
CREATE OR REPLACE VIEW UNPAID_FINES AS
SELECT L.LOAN_ID, L.RESOURCE_ID, L.COPY_NO, L.MEMBER_ID,
L.CHECKOUT_DATE_TIME, L.LOAN_STATUS, L.FINE_AMOUNT, M.FIRST_NAME,
M.LAST_NAME
FROM LOAN_ACTIVE L JOIN MEMBER M ON L.MEMBER_ID = M.MEMBER_ID
WHERE L.FINE_AMOUNT >= 1
ORDER BY L.FINE_AMOUNT DESC;
```

Output:

```
SELECT *
FROM UNPAID_FINES;
```



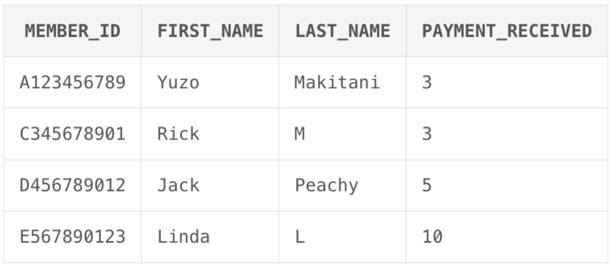
LOAN_ID	RESOURCE_ID	COPY_NO	MEMBER_ID	CHECKOUT_DATE_TIME	LOAN_STATUS	FINE_AMOUNT	FIRST_NAME	LAST_NAME
L0	890123456H	0	B234567890	01-OCT-23 09.00.00.000000 AM	1	45	Jeff	Wang
L1	890123456H	1	D456789012	01-OCT-23 10.00.00.000000 AM	1	45	Jack	Peachy
L7	123456789A	2	B234567890	06-NOV-23 09.00.00.000000 AM	1	10	Jeff	Wang
L9	456789012D	0	C345678901	19-NOV-23 08.00.08.000000 AM	1	8	Rick	M
L11	567890123E	1	D456789012	21-NOV-23 09.00.00.000000 AM	1	7	Jack	Peachy
L10	456789012D	1	D456789012	20-NOV-23 01.19.20.000000 PM	1	5	Jack	Peachy

Fines Paid
The ‘Fines Paid’ view outputs the total fine amount paid by each member of the library; all members with a Payment_Received attribute that is greater than or equal to one.

This view functions as a tool for helping record fine revenue and has applications for helping finance staff with financial planning. This view can be updated periodically (every week, month or quarter etc.) depending on preference and the totals can be recorded manually. Over time, these recordings can be compared to calculate changes in fine amounts, both overall and per member, over time. Therefore, finance staff will have access to information such as monthly fine amounts, month-on-month percentage changes in fine collection, year-on-year percentage changes in fine collection and future fine collection modeling.

```
CREATE OR REPLACE VIEW FINES_PAID AS
SELECT M.MEMBER_ID, M.FIRST_NAME, M.LAST_NAME, M.PAYMENT_RECEIVED
FROM MEMBER M
WHERE M.PAYMENT_RECEIVED >= 1
ORDER BY M.MEMBER_ID;
```

Output:
```
SELECT *
FROM FINES_PAID;
```



MEMBER_ID	FIRST_NAME	LAST_NAME	PAYMENT_RECEIVED
A123456789	Yuzo	Makitani	3
C345678901	Rick	M	3
D456789012	Jack	Peachy	5
E567890123	Linda	L	10

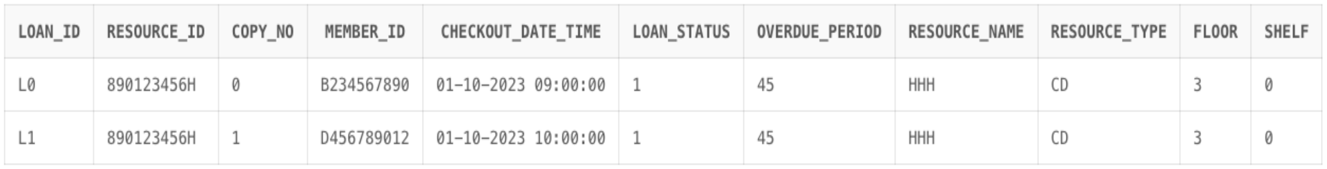
<u>Lost Resources</u>
The ‘Lost Resources’ view outputs all overdue loans which have remained overdue for a period long enough for library staff to consider the resources lost.

If a library member loses or fails to return a resource to the library, it needs to be replaced so it can be re-made available to the other library members. It is important lost resources are replaced so that the library can more successfully fulfill its goal of efficiently circulating resources among members. The amount of time allowed to elapse before a resource is considered ‘lost’ adopted by many prominent libraries, including Queen Mary University of London Library, is 6 weeks (35 days) (Queen Mary University of London, 2023).

The ‘Lost Resources’ view provides library staff system users with the Resource_ID, Copy_No, Resource_Name, Resource_Type, Floor_No and Shelf_No of a lost resource copy so that the library staff in charge of repurchasing can, a) correctly identify the resource copy that is in need of replacement, and b) put the resource copy back in its correct position within the library once it has been repurchased. Once the resource is repurchased and put back, it is assumed that the library staff member will firstly change the Loan_Status from active to inactive; a process which accordingly deletes the loan from Loan Active (though it is kept in Loan Archive), and thus means the loan will no longer appear in the Lost Resources view. Secondly, the staff member will mark the replaced resource copy as available using the Availability attribute in the Copy entity.

```
CREATE OR REPLACE VIEW LOST_RESOURCES AS
SELECT LA.LOAN_ID, LA.RESOURCE_ID, LA.COPY_NO, LA.MEMBER_ID,
TO_CHAR(LA.CHECKOUT_DATE_TIME, 'DD-MM-YYYY HH24:MI:SS') AS
CHECKOUT_DATE_TIME,
   LA.LOAN_STATUS,
   EXTRACT(DAY FROM (SYSTIMESTAMP - LA.CHECKOUT_DATE_TIME)) -
LR.LOAN_PERIOD AS OVERDUE_PERIOD,
   LR.RESOURCE_NAME, LR.RESOURCE_TYPE, C.FLOOR, C.SHELF
FROM LOAN_ACTIVE LA JOIN LIB_RESOURCE LR ON LA.RESOURCE_ID =
LR.RESOURCE_ID
JOIN COPY C ON LA.RESOURCE_ID = C.RESOURCE_ID AND LA.COPY_NO =
C.COPY_NO
WHERE LA.LOAN_STATUS = 1 AND SYSTIMESTAMP - LA.CHECKOUT_DATE_TIME >=
INTERVAL '35' DAY;
```

Output:
```
SELECT *
FROM LOST_RESOURCES;
```



LOAN_ID	RESOURCE_ID	COPY_NO	MEMBER_ID	CHECKOUT_DATE_TIME	LOAN_STATUS	OVERDUE_PERIOD	RESOURCE_NAME	RESOURCE_TYPE	FLOOR	SHELF
L0	890123456H	0	B234567890	01-10-2023 09:00:00	1	45	HHH	CD	3	0
L1	890123456H	1	D456789012	01-10-2023 10:00:00	1	45	HHH	CD	3	0

Resource Analytics View
The ‘Resource Analytics’ view outputs analytics related to the number of loans and copies of a resource.

This view functions as an analysis tool through which library staff, especially those responsible for purchasing resources, can check which resources are in highest demand. The resources which are most in demand will have the highest values of Loans_Per_Copy. Purchasing extra copies of these resources which are most in demand would likely increase the speed and efficiency of resource circulation among library members. Hence, Loans_Per_Copy serves as a useful analytic metric for the library staff database users to have available in view format.

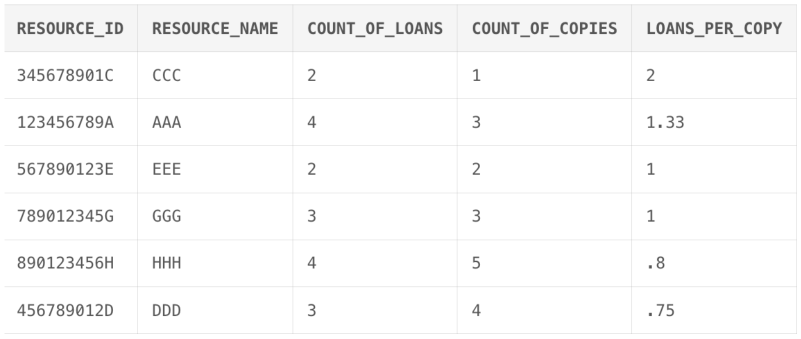
This data is taken from Loan Archive rather than Loan Active because Loan Archive has holistic coverage of loans both active and inactive; Loan Active only has active loans. Library-only resources cannot be loaned out and therefore will not appear in this view.

```
CREATE OR REPLACE VIEW RESOURCE_ANALYTICS AS
SELECT
   LA.RESOURCE_ID,
   LR.RESOURCE_NAME,
   LA.COUNT_OF_LOANS,
   C.COUNT_OF_COPIES,
   CASE
      WHEN C.COUNT_OF_COPIES <> 0 THEN ROUND(LA.COUNT_OF_LOANS /
C.COUNT_OF_COPIES, 2)
      ELSE NULL -- Handle division by 0
   END AS LOANS_PER_COPY
FROM
   (SELECT RESOURCE_ID, COUNT(RESOURCE_ID) AS COUNT_OF_LOANS
   FROM LOAN_ARCHIVE
   GROUP BY RESOURCE_ID) LA
LEFT JOIN
   (SELECT RESOURCE_ID, COUNT(COPY_NO) AS COUNT_OF_COPIES
   FROM COPY
   GROUP BY RESOURCE_ID) C
ON LA.RESOURCE_ID = C.RESOURCE_ID
LEFT JOIN LIB_RESOURCE LR
ON
   LA.RESOURCE_ID = LR.RESOURCE_ID
```

```
ORDER BY LOANS_PER_COPY DESC;
```

Output:

```
SELECT *
FROM RESOURCE_ANALYTICS;
```



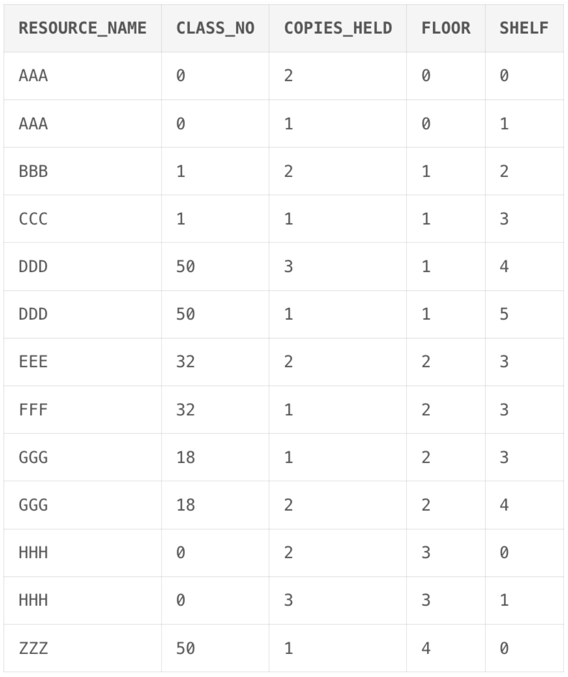
RESOURCE_ID	RESOURCE_NAME	COUNT_OF_LOANS	COUNT_OF_COPIES	LOANS_PER_COPY
345678901C	CCC	2	1	2
123456789A	AAA	4	3	1.33
567890123E	EEE	2	2	1
789012345G	GGG	3	3	1
890123456H	HHH	4	5	.8
456789012D	DDD	3	4	.75

Section 2d. Twelve SQL Queries with Output Strings

All the queries were tested on November 29th.

-- 1. List all resources contained in the library. Output its class number, how many copies of it are held by the library, and the location of the resource.

```
SELECT LR.RESOURCE_NAME,
       LR.CLASS_NO,
       COUNT(C.COPY_NO) AS COPIES_HELD,
       C.FLOOR,
       C.SHELF
FROM LIB_RESOURCE LR
LEFT JOIN COPY C ON LR.RESOURCE_ID = C.RESOURCE_ID
GROUP BY LR.RESOURCE_NAME, LR.CLASS_NO, C.FLOOR, C.SHELF
ORDER BY LR.RESOURCE_NAME;
```



RESOURCE_NAME	CLASS_NO	COPIES_HELD	FLOOR	SHELF
AAA	0	2	0	0
AAA	0	1	0	1
BBB	1	2	1	2
CCC	1	1	1	3
DDD	50	3	1	4
DDD	50	1	1	5
EEE	32	2	2	3
FFF	32	1	2	3
GGG	18	1	2	3
GGG	18	2	2	4
HHH	0	2	3	0
HHH	0	3	3	1
ZZZ	50	1	4	0

-- 2. List all the student and staff members of the library.

```
SELECT LAST_NAME, FIRST_NAME, MEMBER_TYPE
FROM MEMBER
ORDER BY MEMBER_TYPE, LAST_NAME;
```

LAST_NAME	FIRST_NAME	MEMBER_TYPE
M	Rick	staff
Makitani	Yuzo	staff
Wang	Jeff	staff
J	Steve	student
L	Wallis	student
W	Lia	student
Wang	Joel	student
H	Denise	student_staff
L	Linda	student_staff
Peachy	Jack	student_staff

-- 3. List all current reservations. Check if the reserved item is passed on to the next person when the reservation becomes unavailable.

```
SELECT *
FROM RESERVATION;
```

RESERVATION_ID	RESERVATION_DATE_TIME	RESERVATION_STATUS	REMAINING_CHANCES	MEMBER_ID	RESOURCE_ID
R1	20-NOV-23 08.00.00.000000 AM	0	3	F678901234	345678901C
R2	21-NOV-23 09.00.00.000000 AM	1	3	G789012345	345678901C
R3	22-NOV-23 10.00.00.000000 AM	0	0	A123456789	789012345G
R4	26-NOV-23 01.00.00.000000 PM	1	3	E567890123	123456789A
R5	27-NOV-23 02.00.00.000000 PM	1	2	A123456789	890123456H

-- Select all members who made a reservation for the resource '345678901C':

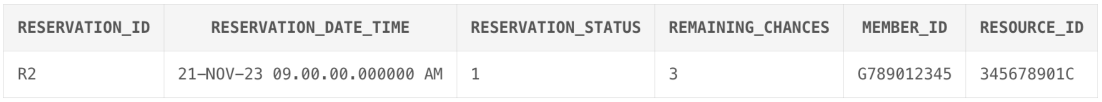
```
SELECT *
FROM RESERVATION
WHERE
RESOURCE_ID = '345678901C';
```

RESERVATION_ID	RESERVATION_DATE_TIME	RESERVATION_STATUS	REMAINING_CHANCES	MEMBER_ID	RESOURCE_ID
R1	20-NOV-23 08.00.00.000000 AM	0	3	F678901234	345678901C
R2	21-NOV-23 09.00.00.000000 AM	1	3	G789012345	345678901C

-- Select only the member first in line for the reservation

```
SELECT *
```

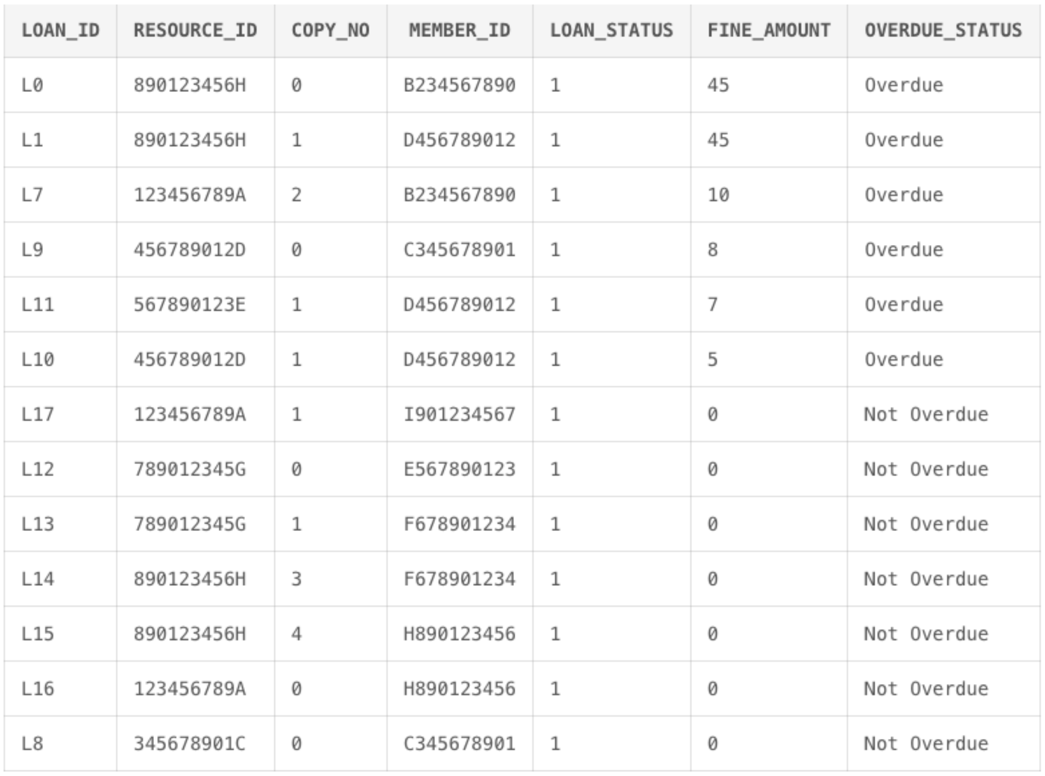
```
FROM RESERVATION
WHERE
RESOURCE_ID = '345678901C' AND
RESERVATION_DATE_TIME = (
          SELECT MIN(RESERVATION_DATE_TIME)
          FROM RESERVATION
          WHERE RESOURCE_ID = '345678901C' AND RESERVATION_STATUS != 0
   );
```



RESERVATION_ID	RESERVATION_DATE_TIME	RESERVATION_STATUS	REMAINING_CHANCES	MEMBER_ID	RESOURCE_ID
R2	21-NOV-23 09.00.00.000000 AM	1	3	G789012345	345678901C

-- 4. List all active loans and their overdue status.

```
SELECT LOAN_ID, RESOURCE_ID, COPY_NO, MEMBER_ID, LOAN_STATUS,
FINE_AMOUNT,
CASE
WHEN LOAN_STATUS = 1 AND FINE_AMOUNT > 0
THEN 'Overdue'
ELSE 'Not Overdue'
END AS OVERDUE_STATUS
FROM LOAN_ACTIVE
ORDER BY FINE_AMOUNT DESC;
```



LOAN_ID	RESOURCE_ID	COPY_NO	MEMBER_ID	LOAN_STATUS	FINE_AMOUNT	OVERDUE_STATUS
L0	890123456H	0	B234567890	1	45	Overdue
L1	890123456H	1	D456789012	1	45	Overdue
L7	123456789A	2	B234567890	1	10	Overdue
L9	456789012D	0	C345678901	1	8	Overdue
L11	567890123E	1	D456789012	1	7	Overdue
L10	456789012D	1	D456789012	1	5	Overdue
L17	123456789A	1	I901234567	1	0	Not Overdue
L12	789012345G	0	E567890123	1	0	Not Overdue
L13	789012345G	1	F678901234	1	0	Not Overdue
L14	890123456H	3	F678901234	1	0	Not Overdue
L15	890123456H	4	H890123456	1	0	Not Overdue
L16	123456789A	0	H890123456	1	0	Not Overdue
L8	345678901C	0	C345678901	1	0	Not Overdue

```
-- 5. List all previous loans by order of popularity.

SELECT LR.RESOURCE_NAME, COUNT(L.RESOURCE_ID)
FROM LOAN_ARCHIVE L
LEFT JOIN LIB_RESOURCE LR
    ON L.RESOURCE_ID = LR.RESOURCE_ID
GROUP BY LR.RESOURCE_NAME
ORDER BY COUNT(L.RESOURCE_ID) DESC;
```

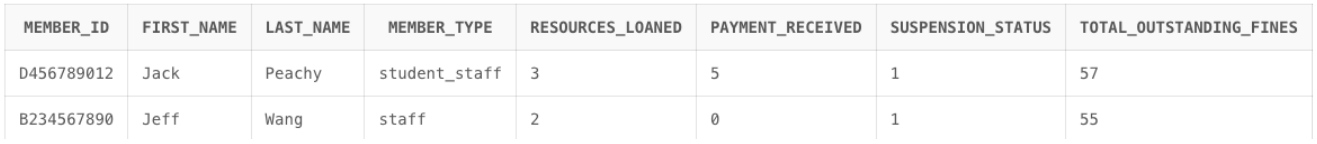
RESOURCE_NAME	COUNT(L.RESOURCE_ID)
HHH	4
AAA	4
DDD	3
GGG	3
CCC	2
EEE	2

```
-- 6. List all details of fines owed by members in order of fine amount.

SELECT M.MEMBER_ID, M.FIRST_NAME, M.LAST_NAME, M.MEMBER_TYPE,
L.RESOURCE_ID, L.CHECKOUT_DATE_TIME, L.LOAN_STATUS, L.FINE_AMOUNT
FROM LOAN_ACTIVE L
LEFT JOIN MEMBER M
ON L.MEMBER_ID = M.MEMBER_ID
WHERE L.FINE_AMOUNT > 0 AND M.SUSPENSION_STATUS=1
ORDER BY FINE_AMOUNT DESC, M.MEMBER_ID DESC;
```

MEMBER_ID	FIRST_NAME	LAST_NAME	MEMBER_TYPE	RESOURCE_ID	CHECKOUT_DATE_TIME	LOAN_STATUS	FINE_AMOUNT
D456789012	Jack	Peachy	student_staff	890123456H	01-OCT-23 10.00.00.000000 AM	1	45
B234567890	Jeff	Wang	staff	890123456H	01-OCT-23 09.00.00.000000 AM	1	45
B234567890	Jeff	Wang	staff	123456789A	06-NOV-23 09.00.00.000000 AM	1	10
D456789012	Jack	Peachy	student_staff	567890123E	21-NOV-23 09.00.00.000000 AM	1	7
D456789012	Jack	Peachy	student_staff	456789012D	20-NOV-23 01.19.20.000000 PM	1	5

```
-- 7. List library members who are suspended due to overdue loans or unpaid fines.

SELECT M.MEMBER_ID, M.FIRST_NAME, M.LAST_NAME, M.MEMBER_TYPE,
M.RESOURCES_LOANED, M.PAYMENT_RECEIVED, M.SUSPENSION_STATUS,
SUM(L.FINE_AMOUNT) AS TOTAL_OUTSTANDING_FINES
FROM MEMBER M
RIGHT JOIN LOAN_ACTIVE L
```

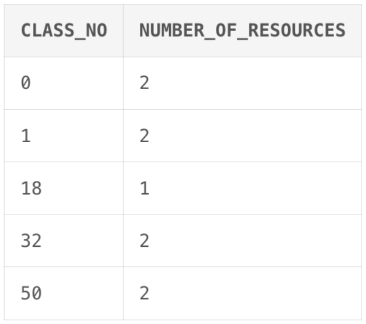
```
ON L.MEMBER_ID = M.MEMBER_ID
WHERE M.SUSPENSION_STATUS = 1
GROUP BY M.MEMBER_ID, M.FIRST_NAME, M.LAST_NAME, M.MEMBER_TYPE,
M.RESOURCES_LOANED, M.PAYMENT_RECEIVED, M.SUSPENSION_STATUS
ORDER BY SUM(L.FINE_AMOUNT) DESC;
```



MEMBER_ID	FIRST_NAME	LAST_NAME	MEMBER_TYPE	RESOURCES_LOANED	PAYMENT_RECEIVED	SUSPENSION_STATUS	TOTAL_OUTSTANDING_FINES
D456789012	Jack	Peachy	student_staff	3	5	1	57
B234567890	Jeff	Wang	staff	2	0	1	55

-- 8. List how many resources are in each class

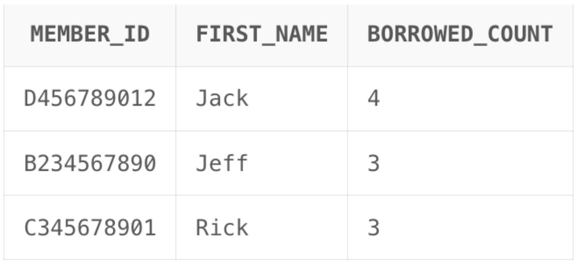
```
SELECT CLASS_NO, COUNT(RESOURCE_ID) AS NUMBER_OF_RESOURCES
FROM LIB_RESOURCE
WHERE CLASS_NO IS NOT NULL
GROUP BY CLASS_NO
ORDER BY CLASS_NO;
```



CLASS_NO	NUMBER_OF_RESOURCES
0	2
1	2
18	1
32	2
50	2

-- 9. List the Top 3 borrowers of all time:

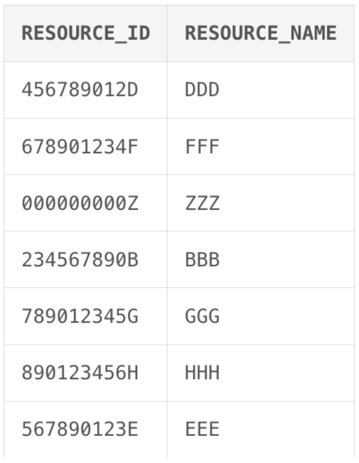
```
SELECT M.MEMBER_ID, M.FIRST_NAME, COUNT(L.LOAN_ID) AS BORROWED_COUNT
FROM MEMBER M
JOIN LOAN_ARCHIVE L ON M.MEMBER_ID = L.MEMBER_ID
GROUP BY M.MEMBER_ID, M.FIRST_NAME
ORDER BY BORROWED_COUNT DESC
FETCH FIRST 3 ROWS ONLY;
```



MEMBER_ID	FIRST_NAME	BORROWED_COUNT
D456789012	Jack	4
B234567890	Jeff	3
C345678901	Rick	3

-- 10. List all the resources which have available copies:

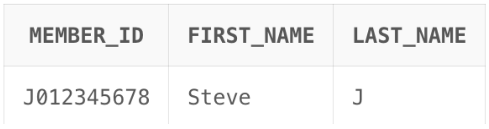
```
SELECT DISTINCT R.RESOURCE_ID, R.RESOURCE_NAME
FROM LIB_RESOURCE R
JOIN COPY C ON R.RESOURCE_ID = C.RESOURCE_ID
WHERE C.AVAILABILITY = 1;
```



RESOURCE_ID	RESOURCE_NAME
456789012D	DDD
678901234F	FFF
000000000Z	ZZZ
234567890B	BBB
789012345G	GGG
890123456H	HHH
567890123E	EEE

-- 11. List all inactive members who have never loaned or reserved any resources:

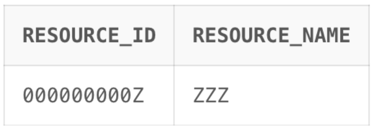
```
SELECT M.MEMBER_ID, M.FIRST_NAME, M.LAST_NAME
FROM MEMBER M
WHERE M.MEMBER_ID NOT IN (
   SELECT DISTINCT MEMBER_ID FROM LOAN_ARCHIVE
   UNION
   SELECT DISTINCT MEMBER_ID FROM RESERVATION
);
```



MEMBER_ID	FIRST_NAME	LAST_NAME
J012345678	Steve	J

-- 12. List the resources that have never been borrowed:

```
SELECT RESOURCE_ID, RESOURCE_NAME
FROM LIB_RESOURCE
WHERE RESOURCE_ID NOT IN(
   SELECT DISTINCT L.RESOURCE_ID
   FROM LOAN_ARCHIVE L
   JOIN COPY C ON L.COPY_NO = C.COPY_NO AND L.RESOURCE_ID = C.RESOURCE_ID
) AND LOAN_PERIOD > 0;
```



RESOURCE_ID	RESOURCE_NAME
000000000Z	ZZZ

Section 2e. Testing Table Constraints (Optional)

Staff active loans may not exceed 10

INSERT INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED, PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME) VALUES ('A1', 'staff', 11, 0, 0, 'a', 'A')

Student_Staff active loans may not exceed 10

INSERT INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED, PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME) VALUES ('B2', 'student_staff', 11, 0, 0, 'b', 'B')

Student active loans may not exceed 5

INSERT INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED, PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME) VALUES ('C3', 'student', 6, 0, 0, 'c', 'C')

Suspension Status can only be 0 or 1

INSERT INTO MEMBER (MEMBER_ID, MEMBER_TYPE, RESOURCES_LOANED, PAYMENT_RECEIVED, SUSPENSION_STATUS, FIRST_NAME, LAST_NAME) VALUES ('D4', 'student', 3, 0, 2, 'd', 'D')

Availability can only be 0 or 1

INSERT INTO COPY (COPY_NO, RESOURCE_ID, FLOOR, SHELF, AVAILABILITY) VALUES (0, '1A', 0, 0, 2);

Loan Status in Loan Archive and Loan Active can only be 0 or 1

INSERT INTO LOAN_ARCHIVE (LOAN_ID, MEMBER_ID, CHECKOUT_DATE_TIME, LOAN_STATUS, FINE_AMOUNT, COPY_NO, RESOURCE_ID) VALUES (23, 'B234567890', TIMESTAMP '2023-12-01 00:00:00', 2, 0, 0, '123456789A')

Reservation Status can only be 0 or 1

INSERT INTO RESERVATION (RESERVATION_ID, RESERVATION_DATE_TIME, RESERVATION_STATUS, REMAINING_CHANCES, MEMBER_ID, RESOURCE_ID) VALUES ('R16', TIMESTAMP '2023-11-01 00:00:00', 2, 0, 'A123456789', '678901234F')

Remaining Chances may not exceed 3

```
INSERT INTO RESERVATION (RESERVATION_ID, RESERVATION_DATE_TIME,
RESERVATION_STATUS, REMAINING_CHANCES, MEMBER_ID, RESOURCE_ID) VALUES
('R17', TIMESTAMP '2023-11-02 00:00:30', 0, 4, 'B234567890', '789012345G')
```

Bibliography

Queen Mary University of London (2023) *Lost or damaged books - Library Services.* [online] www.qmul.ac.uk, Available at: https://www.qmul.ac.uk/library/using-library-services/borrowing-basics/lostdamaged-or-overdue-materials/ [Accessed 22 Nov. 2023].

Thibodeaux, C. (2023). *20 Surprising Facts About The British Library*. [online] Facts.net. Available at: https://facts.net/world/landmarks/20-surprising-facts-about-the-british-library/ [Accessed 29 Nov. 2023].