Objective: You have designed the logical blueprint for your database. Now, create a physical database based on this logical design and populate it with meaningful data. Although the deliverable (artifact) for this part of the project will be a Canvas submission, you must ensure you have a fully functional database. This database should be rigorously tested to verify that it meets all the requirements outlined in your project proposal.

Introduction [5 points]. *Project Overview:* Briefly reiterate the purpose and main functionalities of your database, maintaining consistency with your previous project parts. *Scope***:** Re-emphasize the boundaries of your project, clarifying included and excluded data and functionalities. *Glossary***:** Update your glossary with any new terms or concepts relevant to relational database design. The introduction section is for continuity; the italic parts are subsections.

Project Overview: The purpose of this database is to store a collection of different books, magazines, movies able to be rented out by different users.

- ***Scope:***
    - *The purpose of a Library Management System is to make a useful, scalable, and educational tool that works like a real library. As part of the project, we will be able to work directly with all aspects of database development, from conceptual modeling to logical design to physical implementation. We will also add ways to make sure that borrowing rules are followed and useful reports will be made to help with operational decisions, collection growth, and member participation. At the end of the day, this project will be both a useful way to learn and show how well-designed database systems can help solve difficult information management issues.*

- ***Glossary:***
    - *ISBN: International Standard Book Number; is a unique, 13-digit numerical identifier for a specific book edition and format, used by libraries, bookstores, and online retailers to catalog, track, and sell books globally.*
    - *SQL: Structured Query Language*
    - *Functional Requirement: specify what a system must do, detailing the specific actions, functions, and features it needs to perform to meet user and business needs*
    - *Non-Functional Requirement: specify what a system must do, detailing the specific actions, functions, and features it needs to perform to meet user and business needs*

Choose Your Platform [5 points]: You can use the EECS Cycle servers with MariaDB, or if your team prefers, select another database product (like MySQL, PostgreSQL, or SQL Server). Be sure to explain your choice in your report. What factors influenced your decision? Familiarity with the system? Specific features? Potential limitations?

- *Platform Choice: SQLite was chosen for its simplicity, local hosting capability, and compatibility with small-scale academic projects. It does not require a server setup, making it ideal for rapid testing and debugging during development.*

Create Your Database: Create a physical database schema using your DDL scripts. Ensure all tables are correctly created with appropriate constraints (primary keys, foreign keys, etc.). Organize all SQL scripts (DDL, data population, queries, reports) clearly, name them descriptively, and include comments explaining their functions.

Print Your Physical Schema [10 points]: Print the SQL DDL statements that you used to create the physical schema.

```
/*

NAME: EECS 447 PROJECT (DDL)

DESCRIPTION: DDL OF DATABASE SYSTEM FOR LIBRARY MANAGEMENT

CONTRIBUTORS: JAIDEN GREEN, SHERO BAIG, LUIS, ERIC, AMRIT

START DATE: OCTOBER 22, 2025

LAST UPDATED: November 4, 2025

*/


--Creates Library User Entity Table: Jaiden

CREATE TABLE LIBUSER(

UID CHAR(8) not null primary key,      --Identification Number for Users

Fname VARCHAR(20) not null,            --First name of the user

Lname VARCHAR(20) not null,            --Last name of the user

Phone VARCHAR(15) not null,            --Phone number of the user

Email VARCHAR(50) not null,            --Email address of the user

HouseNum VARCHAR(10) not null,         --House number of the user

City VARCHAR(20) not null,             --City of the user

State VARCHAR(20) not null,            --State of the user

ZIP CHAR(5) not null,                  --ZIP code of the user

Street VARCHAR(50) not null,           --Street of the user

AptNum VARCHAR(10)                     --Apartment number of the user

);


--CREATES CLIENT SPECIALIZATION TABLE: Jaiden

CREATE TABLE Client(

UID CHAR(8) not null PRIMARY KEY,          --Identification number for User

AcctStanding INT,                          --Stores how many items are loaned(should not be more than 3)

MemStart DATE,                             --Date of membership start

FOREIGN KEY (UID) REFERENCES LIBUSER(UID)
```

```
);

--CREATES LIBRARIAN SPECIALIZATION TABLE: Jaiden
CREATE TABLE Librarian(
UID CHAR(8) not null PRIMARY KEY,               --Identification number for User
Salary Float(10,2),                             --Salary of the librarian
FOREIGN KEY (UID) REFERENCES LIBUSER(UID)
);

--Moved the Items Table above Reserves to make sure Reserves is referencing Items Correctly; Jaiden
--CREATES ITEMS ENTITY TABLE: SHERO
CREATE TABLE IF NOT EXISTS Items(
CopyID CHAR(8) NOT NULL PRIMARY KEY, --Identification Number for the Copy, primary key of Items entity
Title VARCHAR(50) NOT NULL, -- Title of the Copy with up to 50 characters allowed, cant be null
Genre VARCHAR(15) NOT NULL, -- Genre of the Copy with up to 15 characters allowed, cant be null
ReleaseDate DATE NOT NULL, -- Release date of the Copy, cant be null
Availability BOOLEAN NOT NULL, -- Availability of Copy, either Availabile or not (True or False), cant be null
NumofCheckouts SMALLINT, -- Tells the number of times the copy has been checked out, smallint used cause it will be small whole numbers
Duedate DATE -- Date when the copy is due
);
--CREATES BOOKS SPECIALIZATION TABLE: SHERO
CREATE TABLE IF NOT EXISTS Books(
CopyID CHAR(8) NOT NULL PRIMARY KEY, --Identification Number for the Copy, primary key of books specialization
ISBN CHAR(13) NOT NULL, --ISBN Number for books with a fixed amount of 13 characters, cant be null
Author VARCHAR(50) NOT NULL, -- Author's name with up to 50 characters allowed, cant be null
FOREIGN KEY (CopyID) REFERENCES Items(CopyID) -- foreign key is the copyID cause it is referenced from Items
);

--CREATES MAGAZINES SPECIALIZATION TABLE: SHERO
CREATE TABLE IF NOT EXISTS Magazines(
CopyID CHAR(8) NOT NULL PRIMARY KEY, --Identification Number for the Copy, primary key of magazines specialization with a fixed amount of 8 characters, and cant be null
IssueNum CHAR(6) NOT NULL, --Issue number of the magazine with a fixed amount of 6 characters, and cant be null
Publisher VARCHAR(50) NOT NULL, --Publisher of the magazine that can go up to 50 characters
FOREIGN KEY (CopyID) REFERENCES Items(CopyID) -- foreign key is the copyID cause it is referenced from Items
);

--CREATES MOVIES SPECIALIZATION TABLE: SHERO
CREATE TABLE IF NOT EXISTS Movies(
CopyID CHAR(8) NOT NULL PRIMARY KEY, --Identification Number for the Copy, primary key of movies specialization with a fixed amount of 8 characters, and cant be null
Rating CHAR(4) NOT NULL, --Rating on rotten tomatoes for the movie, with a fixed amount of 4 characters, cant be null
Director VARCHAR(50) NOT NULL, --Director of the movie, can go up to 50 characters, cant be null
FOREIGN KEY (CopyID) REFERENCES Items(CopyID)
);

--CREATES RESERVES RELATINSHIP TABLE: Jaiden
CREATE TABLE Reserves(
CopyID CHAR(8) not null,            --Identification number for Items
UID CHAR(8) not null,                           --Identification number for User
```

```
ReservedAt DATE,

FOREIGN KEY (CopyID) REFERENCES Items(CopyID),

FOREIGN KEY (UID) REFERENCES Client(UID)

);


-- CREATES FEE ENTITY TABLE: LUIS

CREATE TABLE IF NOT EXISTS Fee(

FeeID CHAR(8) NOT NULL PRIMARY KEY,                --Identification Number for Fee

UID CHAR(8) NOT NULL,                    --Foreign Key of Client'sID

DueDate DATE NOT NULL,                   --Date Fee is Due

PaidAt DATE,                  --Date Fee is Paid

Overdue          BOOLEAN NOT NULL,                  --y/n if fee is overdue

PaidStatus BOOLEAN NOT NULL,          --y/n if the fee is paid

PaymentMethod VARCHAR(15) NOT NULL,      --how was the fee paid

TransactionID CHAR(8) NOT NULL,      --Foreign key to Transaction ID

Amount NUMERIC(12,2) NOT NULL,          --Amount of money due

FOREIGN KEY (UID) REFERENCES Client(UID),

FOREIGN KEY (TransactionID) REFERENCES UserTransaction(TransactionID)

);


-- CREATES RENTS RELATIONSHIP TABLE: LUIS

CREATE TABLE IF NOT EXISTS Rents(

TransactionID CHAR(8) NOT NULL,      --Identifies the checkout/return record this copy is part of

UID CHAR(8) NOT NULL,               --Identification number for User

FOREIGN KEY (TransactionID) REFERENCES UserTransaction(TransactionID),

FOREIGN KEY (UID) REFERENCES Client(UID)

);



--CREATES RENTED RELATIONSHIP TABLE: SHERO

CREATE TABLE IF NOT EXISTS Rented(

TransactionID CHAR(8) NOT NULL PRIMARY KEY, -- Idenitfies the checkout/return record this copy is part of. Up to fixed amount of 8 characters and a primary key and cant be null

CopyID CHAR(8) NOT NULL, -- Identifies the exact physical copy that's being rented. Up to fixed amount of 8 characters and cant be null

FOREIGN KEY (TransactionID) REFERENCES UserTransaction(TransactionID), --Foreign key is transactionid and it is referenced from UserTransaction

FOREIGN KEY (CopyID) REFERENCES Items(CopyID) --Another foreign key is CopyID and it is referenced from Items

);


-- Create UserTransaction table: Amrit

CREATE TABLE IF NOT EXISTS UserTransaction (        -- Creates the UserTransaction table if it doesn't already exist

    TransactionID CHAR(8) PRIMARY KEY,          -- Unique transaction identifier

    LateFee NUMERIC(12,2) DEFAULT 0 CHECK (LateFee >= 0),     -- Late fee amount, must be non-negative

    CopyID CHAR(8) NOT NULL,               -- References the specific item copy being rented

    ClientID CHAR(8) NOT NULL,             -- References the client performing the transaction

    LibrarianID CHAR(8) NOT NULL,           -- References the librarian processing the transaction

    CheckoutDate DATE NOT NULL,             -- Date when the item was checked out

    DueDate DATE NOT NULL,                -- Date when the item is due for return

    ReturnDate DATE,                  -- Date when the item was returned (nullable)

    CHECK (DueDate >= CheckoutDate),          -- Ensures due date is not before checkout date
    CHECK (ReturnDate IS NULL OR ReturnDate >= CheckoutDate),    -- Ensures return date is not before checkout date
    FOREIGN KEY (CopyID) REFERENCES Items(CopyID),              -- Foreign key linking to Items table
    FOREIGN KEY (ClientID) REFERENCES Client(UID),              -- Foreign key linking to Client table
    FOREIGN KEY (LibrarianID) REFERENCES Librarian(UID)         -- Foreign key linking to Librarian table
);                                                              -- Ends table creation

-- Helpful indexes
CREATE INDEX IF NOT EXISTS idx_ut_copy ON UserTransaction(CopyID);      -- Index on CopyID for faster lookups by item
CREATE INDEX IF NOT EXISTS idx_ut_client ON UserTransaction(ClientID);  -- Index on ClientID for efficient client searches
CREATE INDEX IF NOT EXISTS idx_ut_lib ON UserTransaction(LibrarianID);  -- Index on LibrarianID for efficient librarian searches

CREATE TABLE IF NOT EXISTS Assists(
    LibrarianID CHAR(8) NOT NULL, -- References the librarian assisting with the transaction
    TransactionID CHAR(8) NOT NULL, -- References the related transaction
    PRIMARY KEY (LibrarianID, TransactionID), -- Composite primary key (each librarian-transaction pair is unique)
    FOREIGN KEY (LibrarianID) REFERENCES Librarian(UID), -- Foreign key linking to Librarian table
    FOREIGN KEY (TransactionID) REFERENCES UserTransaction(TransactionID) -- Foreign key linking to UserTransaction table
);

-- Creates Pays Table: Eric
CREATE TABLE IF NOT EXISTS Pays (
    FeeID CHAR(8) NOT NULL, -- Identifies the specific fee being paid
    UID CHAR(8) NOT NULL, -- Identifies the user paying the fee
    PRIMARY KEY (FeeID), -- Primary key FeeId uniquely identifies the transaction
    FOREIGN KEY (UID) REFERENCES Client(UID), -- Foreign key linking to Client
    FOREIGN KEY (FeeID) REFERENCES Fee(FeeID) -- Foreign key linking to Fee
    ); -- Ends table creation

CREATE INDEX IF NOT EXISTS idx_assists_lib ON Assists(LibrarianID);      -- Index on LibrarianID for quick lookup by librarian
CREATE INDEX IF NOT EXISTS idx_assists_txn ON Assists(TransactionID);    -- Index on TransactionID for quick lookup by transaction
```

Data Population. Use tools or scripts to create realistic data matching the type of information your database will hold. Utilize public datasets where available, or manually enter data if your project is small. Use bulk data loading utilities for larger datasets. Validate your data for accuracy and consistency with your design.

Printing Table Contents [10 points]. Once the tables are populated, print the contents of each table to provide a quick overview of the table contents, allowing examination of data size and composition. You may use the following SQL command for each table:

```
SELECT * FROM TableName;
```

See additional commands below.

Functionality Testing. Develop SQL queries to store and update data in your database, implement functionalities described in your requirements document (e.g., searching for specific records, generating reports), and create clear, useful reports. Test to ensure:

- Queries and reports produce correct results.
- Database handles unexpected input or edge cases without crashing.
- Database performs well (queries are not excessively slow).

```
/*
NAME: EECS 447 PROJECT: Functionality Testing
DESCRIPTION: Queries/Reports for DATABASE SYSTEM FOR LIBRARY MANAGEMENT
CONTRIBUTORS: JAIDEN GREEN, SHERO BAIG, LUIS, ERIC, AMRIT
START DATE: OCTOBER 29, 2025
LAST UPDATED: NOVEMBER 13, 2025
*/

/*
This file is used to store predetermined queries
Formatted as:

Name: (name of query)
(actual SQL code)

*/

--~~~~~~~~~~~~~~~~~~~~~~~~~~~
--    Searches
--~~~~~~~~~~~~~~~~~~~~~~~~~~
-- Find all available fiction books: Jaiden
-- Name: All fiction books
SELECT Title FROM Items WHERE Genre = 'Fiction' AND Availability = 1;

-- List all clients who currently have overdue items
-- Name: Clients with overdue items
SELECT U.Fname, U.Lname, I.Title, T.DueDate
FROM UserTransaction T
JOIN Client C ON T.ClientID = C.UID
JOIN LIBUSER U ON C.UID = U.UID
JOIN Items I ON T.CopyID = I.CopyID
WHERE T.ReturnDate IS NULL AND T.DueDate < CURRENT_DATE;

-- Display total revenue collected from paid fees
-- Name: Fee Revenue
SELECT SUM(Amount) AS TotalRevenue
FROM FEE
WHERE PaidStatus = TRUE;

-- Show librarians and how many transactions they assisted with
-- Name: Librarian Transaction Assists
SELECT U.Fname, U.Lname, COUNT(A.TransactionID) AS NumTransactionsAssisted
FROM Assists A
JOIN Librarian L ON A.LibrarianID = L.UID
JOIN LIBUSER U ON L.UID = U.UID
GROUP BY U.Fname, U.Lname
ORDER BY NumTransactionsAssisted DESC;

-- Finds clients who do not currently have any reservations (Shero)
-- Name: Clients with No Reservations
SELECT
   U.Fname,
   U.Lname,
```

```
    C.UID
FROM Client C
JOIN LIBUSER U ON C.UID = U.UID
LEFT JOIN Reserves R ON R.UID = C.UID
WHERE R.UID IS NULL;

-- Shows clients who have fees, but no unpaid fees remaining (Shero)
-- Name: Clients with All Fees Paid
SELECT
    U.Fname,
    U.Lname,
    C.UID
FROM Client C
JOIN LIBUSER U ON C.UID = U.UID
WHERE EXISTS (
    SELECT 1
    FROM Fee F
    WHERE F.UID = C.UID
)
AND NOT EXISTS (
    SELECT 1
    FROM Fee F2
    WHERE F2.UID = C.UID
      AND F2.PaidStatus = FALSE
);

--~~~~~~~~~~~~~~~~~~~~~~~~~~
--    Updates
--~~~~~~~~~~~~~~~~~~~~~~~~~~
-- Updates can be done in the dashboard
-- Mark item as rented: Jaiden
-- UPDATE Items SET Availability = 0 WHERE CopyID = 'C0000001';

--~~~~~~~~~~~~~~~~~~~~~~~~~~
--    Joins
--~~~~~~~~~~~~~~~~~~~~~~~~~~

-- Show which client rented which book: Jaiden
-- Name: Client rented books
SELECT U.Fname, U.Lname, I.Title, T.CheckoutDate, T.DueDate
FROM UserTransaction T
JOIN Client C ON T.ClientID = C.UID
JOIN LIBUSER U ON C.UID = U.UID
JOIN Items I ON T.CopyID = I.CopyID;

-- Shows how many unique clients each librarian has helped (Shero)
-- Name: Librarian Client Interaction Count
SELECT
    L.UID AS LibrarianID,
    U.Fname,
    U.Lname,
    COUNT(DISTINCT T.ClientID) AS NumClientsHelped
FROM Librarian L
JOIN LIBUSER U ON L.UID = U.UID
JOIN Assists A ON L.UID = A.LibrarianID
JOIN UserTransaction T ON T.TransactionID = A.TransactionID
GROUP BY L.UID, U.Fname, U.Lname
ORDER BY NumClientsHelped DESC;

--~~~~~~~~~~~~~~~~~~~~~~~~~~
--    Reports
--~~~~~~~~~~~~~~~~~~~~~~~~~~

-- Late returns report: Jaiden
-- Name: Late Returns
SELECT U.Fname || ' ' || U.Lname AS ClientName, I.Title, T.DueDate, T.ReturnDate
FROM UserTransaction T
JOIN Client C ON T.ClientID = C.UID
```

```
JOIN LIBUSER U ON C.UID = U.UID
JOIN Items I ON T.CopyID = I.CopyID
WHERE T.ReturnDate > T.DueDate;

-- Name: Assists Activity
SELECT Fname, Lname, LibrarianID, COUNT(*) AS TransactionsHelped
FROM Assists JOIN LIBUSER ON LibrarianID = UID
GROUP BY LibrarianID
ORDER BY TransactionsHelped DESC;

-- Name: Overdue Fees
SELECT FeeID, UID, DueDate, Overdue, PaidStatus
FROM Fee
WHERE Overdue = TRUE AND PaidStatus = FALSE;

-- Name: Item Availability
SELECT CopyID, Title, Availability
FROM Items
WHERE Availability = TRUE;

-- Name: Checkouts Per Item
SELECT Title, NumofCheckouts
FROM Items
ORDER BY NumofCheckouts DESC

-- Name: Unpaid Fees
SELECT * FROM Fee
WHERE PaidStatus = 0;

-- Name: High Rated Movies
SELECT Title, m.Rating
FROM ITEMS i
JOIN MOVIES m ON i.CopyID = m. CopyID
WHERE m.Rating >= 90;

-- Name: Reserved Items
SELECT l.Fname, l.UID, i.Title
FROM RESERVES  r
JOIN LIBUSER l ON r.UID = l.UID
JOIN ITEMS i ON i.CopyID = r.CopyID

-- Shows how many rentals each client has made
-- Name: Most Active Clients (Rent Count) (Shero)
SELECT
    U.Fname,
    U.Lname,
    R.UID,
    COUNT(*) AS NumRentals
FROM Rents R
JOIN LIBUSER U ON R.UID = U.UID
GROUP BY U.Fname, U.Lname, R.UID
ORDER BY NumRentals DESC;

-- Lists clients who have at least one reservation and at least one rental
-- Name: Clients with Reservations and Rentals  (Shero)
SELECT
    U.Fname,
    U.Lname,
    C.UID
FROM Client C
JOIN LIBUSER U ON C.UID = U.UID
WHERE EXISTS (
    SELECT 1
    FROM Reserves R
    WHERE R.UID = C.UID
)
AND EXISTS (
    SELECT 1
    FROM Rents Rt
    WHERE Rt.UID = C.UID
);

-- Name: Top Circulated Items
SELECT
    Title,
    NumofCheckouts
FROM Items
ORDER BY NumofCheckouts DESC
LIMIT 10;

-- Name: Clients With Overdue Unpaid Fees
SELECT
    U.Fname,
    U.Lname,
    F.FeeID,
    F.Amount,
    F.DueDate
FROM Fee F
JOIN Client C ON F.UID = C.UID
JOIN LIBUSER U ON C.UID = U.UID
WHERE F.Overdue = TRUE
  AND F.PaidStatus = FALSE;

-- Name: Currently Checked Out Items
SELECT
    I.Title,
    T.CopyID,
    U.Fname,
    U.Lname,
    T.CheckoutDate,
    T.DueDate
FROM UserTransaction T
JOIN Items I ON T.CopyID = I.CopyID
JOIN Client C ON T.ClientID = C.UID
JOIN LIBUSER U ON C.UID = U.UID
WHERE T.ReturnDate IS NULL;
```

GitHub Repository Management. Update your GitHub repository with your report, all your scripts, and any other important files.

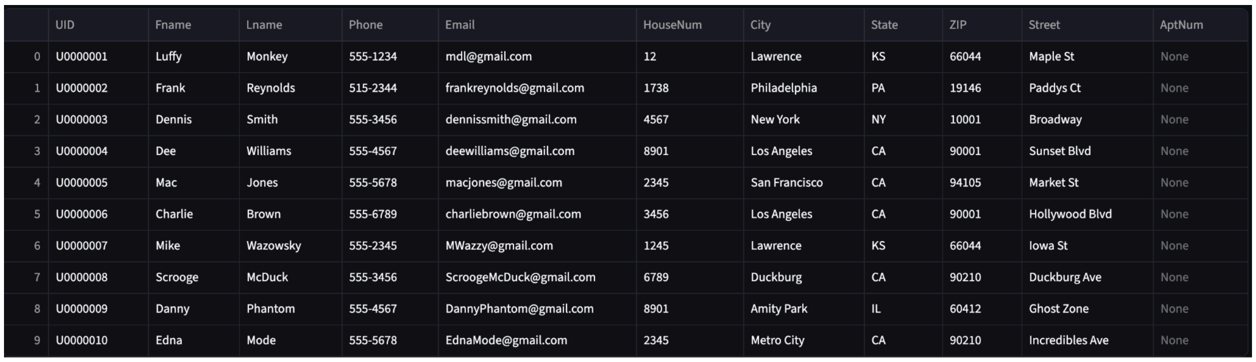
Using `select * from R;` for each table individually is a simple and straightforward.

The above command may not print the table name. The following are steps in MariaDB to print the table names (assume there is a table called `Books`):

```
show tables;    --Displays all table names
select 'table: Books' as ' ';   --Displays name
describe Books;      --Displays meta-data
select * from Books; --Displays table data
```

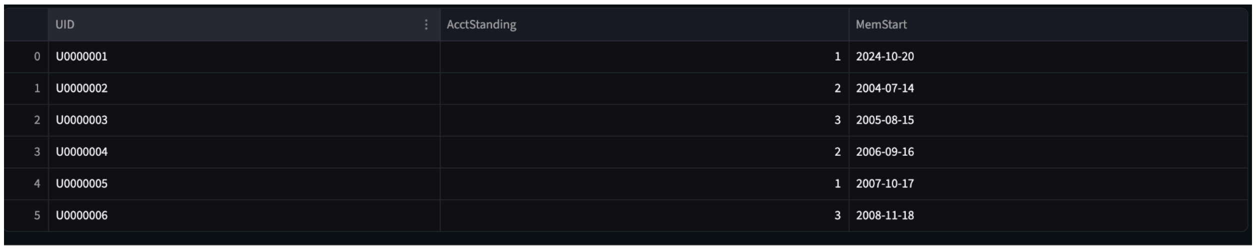
https://github.com/JaidenTGreen/EECS-447-Project/tree/main

Select * From LibUser;



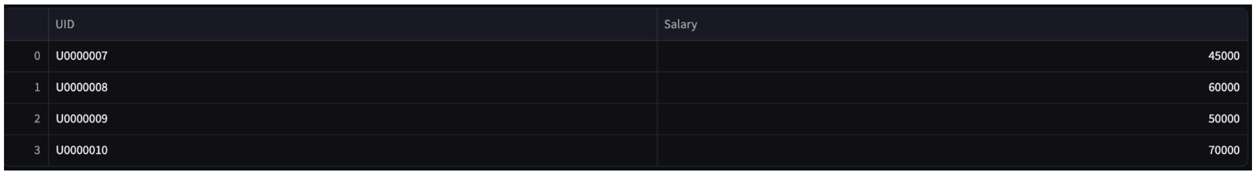
	UID	Fname	Lname	Phone	Email	HouseNum	City	State	ZIP	Street	AptNum
0	U0000001	Luffy	Monkey	555-1234	mdl@gmail.com	12	Lawrence	KS	66044	Maple St	None
1	U0000002	Frank	Reynolds	515-2344	frankreynolds@gmail.com	1738	Philadelphia	PA	19146	Paddys Ct	None
2	U0000003	Dennis	Smith	555-3456	dennissmith@gmail.com	4567	New York	NY	10001	Broadway	None
3	U0000004	Dee	Williams	555-4567	deewilliams@gmail.com	8901	Los Angeles	CA	90001	Sunset Blvd	None
4	U0000005	Mac	Jones	555-5678	macjones@gmail.com	2345	San Francisco	CA	94105	Market St	None
5	U0000006	Charlie	Brown	555-6789	charliebrown@gmail.com	3456	Los Angeles	CA	90001	Hollywood Blvd	None
6	U0000007	Mike	Wazowsky	555-2345	MWazzy@gmail.com	1245	Lawrence	KS	66044	Iowa St	None
7	U0000008	Scrooge	McDuck	555-3456	ScroogeMcDuck@gmail.com	6789	Duckburg	CA	90210	Duckburg Ave	None
8	U0000009	Danny	Phantom	555-4567	DannyPhantom@gmail.com	8901	Amity Park	IL	60412	Ghost Zone	None
9	U0000010	Edna	Mode	555-5678	EdnaMode@gmail.com	2345	Metro City	CA	90210	Incredibles Ave	None

Select * From Client;



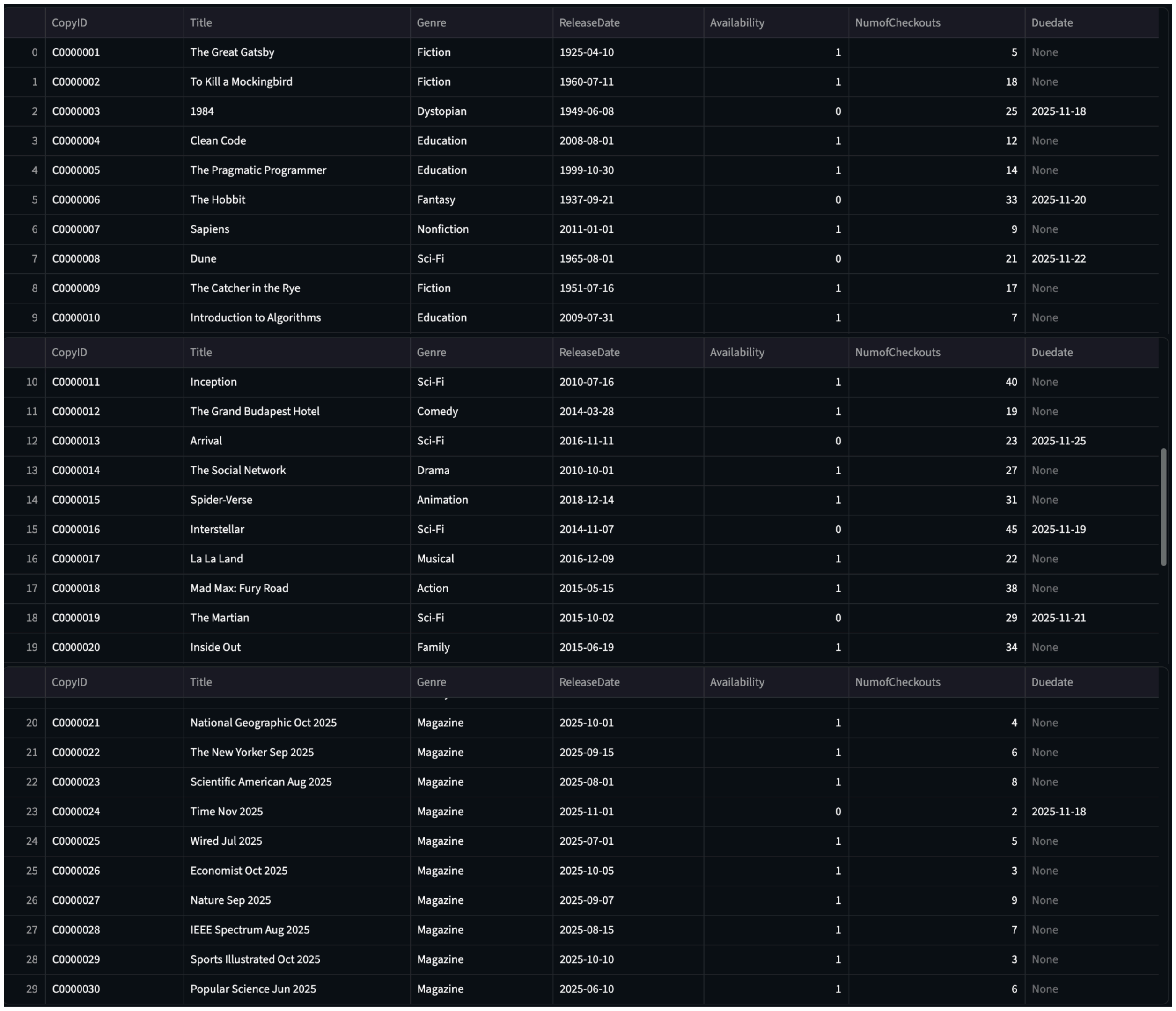
	UID	AcctStanding	MemStart
0	U0000001	1	2024-10-20
1	U0000002	2	2004-07-14
2	U0000003	3	2005-08-15
3	U0000004	2	2006-09-16
4	U0000005	1	2007-10-17
5	U0000006	3	2008-11-18

Select * From Librarian;



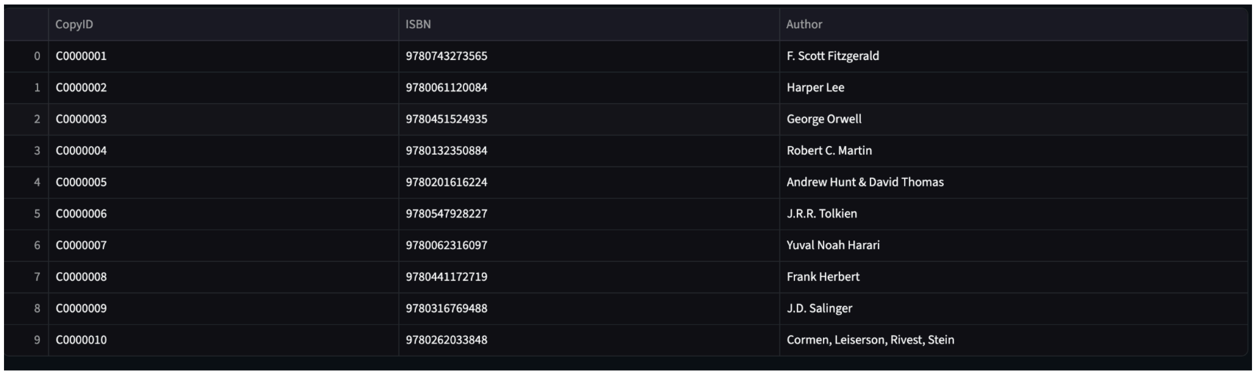
	UID	Salary
0	U0000007	45000
1	U0000008	60000
2	U0000009	50000
3	U0000010	70000

Select * From Items;



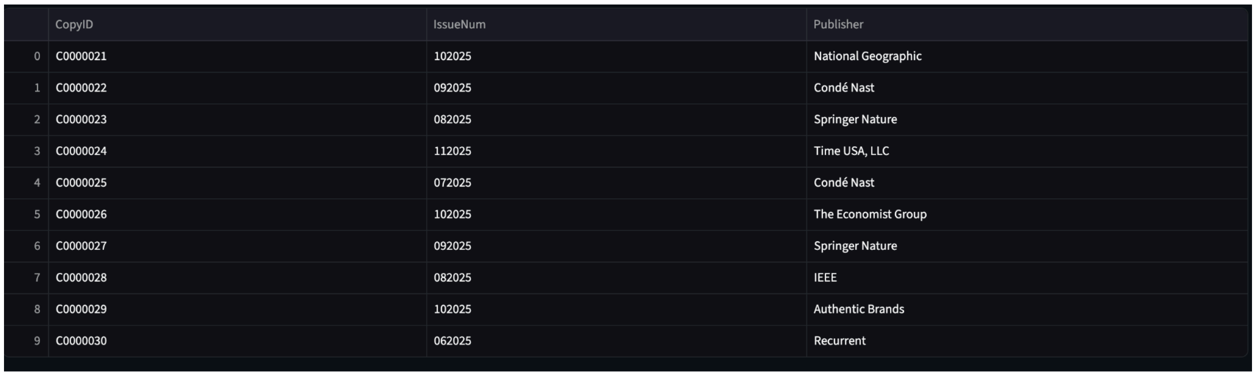
	CopyID	Title	Genre	ReleaseDate	Availability	NumofCheckouts	Duedate
0	C0000001	The Great Gatsby	Fiction	1925-04-10	1	5	None
1	C0000002	To Kill a Mockingbird	Fiction	1960-07-11	1	18	None
2	C0000003	1984	Dystopian	1949-06-08	0	25	2025-11-18
3	C0000004	Clean Code	Education	2008-08-01	1	12	None
4	C0000005	The Pragmatic Programmer	Education	1999-10-30	1	14	None
5	C0000006	The Hobbit	Fantasy	1937-09-21	0	33	2025-11-20
6	C0000007	Sapiens	Nonfiction	2011-01-01	1	9	None
7	C0000008	Dune	Sci-Fi	1965-08-01	0	21	2025-11-22
8	C0000009	The Catcher in the Rye	Fiction	1951-07-16	1	17	None
9	C0000010	Introduction to Algorithms	Education	2009-07-31	1	7	None
10	C0000011	Inception	Sci-Fi	2010-07-16	1	40	None
11	C0000012	The Grand Budapest Hotel	Comedy	2014-03-28	1	19	None
12	C0000013	Arrival	Sci-Fi	2016-11-11	0	23	2025-11-25
13	C0000014	The Social Network	Drama	2010-10-01	1	27	None
14	C0000015	Spider-Verse	Animation	2018-12-14	1	31	None
15	C0000016	Interstellar	Sci-Fi	2014-11-07	0	45	2025-11-19
16	C0000017	La La Land	Musical	2016-12-09	1	22	None
17	C0000018	Mad Max: Fury Road	Action	2015-05-15	1	38	None
18	C0000019	The Martian	Sci-Fi	2015-10-02	0	29	2025-11-21
19	C0000020	Inside Out	Family	2015-06-19	1	34	None
20	C0000021	National Geographic Oct 2025	Magazine	2025-10-01	1	4	None
21	C0000022	The New Yorker Sep 2025	Magazine	2025-09-15	1	6	None
22	C0000023	Scientific American Aug 2025	Magazine	2025-08-01	1	8	None
23	C0000024	Time Nov 2025	Magazine	2025-11-01	0	2	2025-11-18
24	C0000025	Wired Jul 2025	Magazine	2025-07-01	1	5	None
25	C0000026	Economist Oct 2025	Magazine	2025-10-05	1	3	None
26	C0000027	Nature Sep 2025	Magazine	2025-09-07	1	9	None
27	C0000028	IEEE Spectrum Aug 2025	Magazine	2025-08-15	1	7	None
28	C0000029	Sports Illustrated Oct 2025	Magazine	2025-10-10	1	3	None
29	C0000030	Popular Science Jun 2025	Magazine	2025-06-10	1	6	None

Select * From Books;



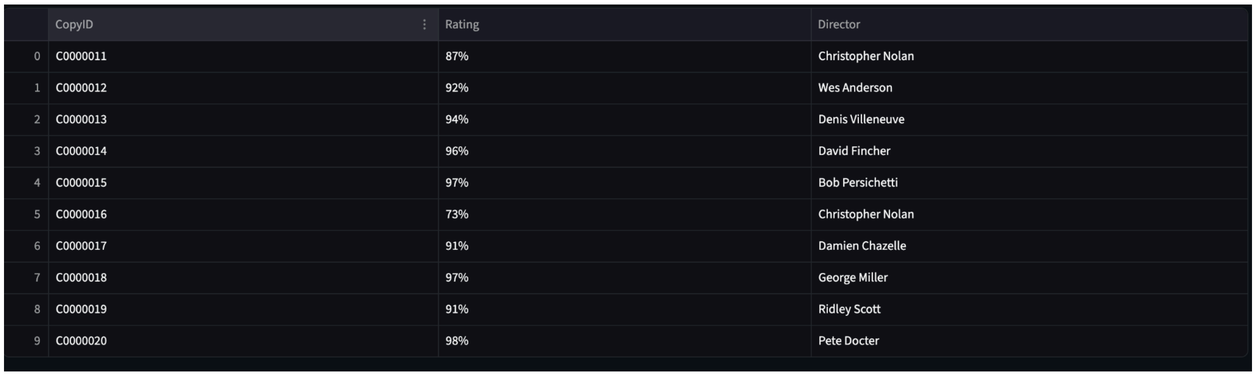
	CopyID	ISBN	Author
0	C0000001	9780743273565	F. Scott Fitzgerald
1	C0000002	9780061120084	Harper Lee
2	C0000003	9780451524935	George Orwell
3	C0000004	9780132350884	Robert C. Martin
4	C0000005	9780201616224	Andrew Hunt & David Thomas
5	C0000006	9780547928227	J.R.R. Tolkien
6	C0000007	9780062316097	Yuval Noah Harari
7	C0000008	9780441172719	Frank Herbert
8	C0000009	9780316769488	J.D. Salinger
9	C0000010	9780262033848	Cormen, Leiserson, Rivest, Stein

Select * From Magazines;



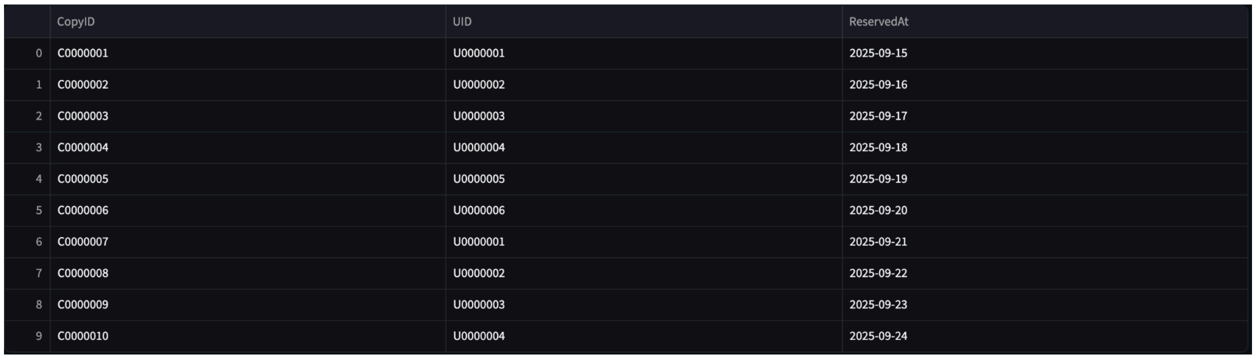
	CopyID	IssueNum	Publisher
0	C0000021	102025	National Geographic
1	C0000022	092025	Condé Nast
2	C0000023	082025	Springer Nature
3	C0000024	112025	Time USA, LLC
4	C0000025	072025	Condé Nast
5	C0000026	102025	The Economist Group
6	C0000027	092025	Springer Nature
7	C0000028	082025	IEEE
8	C0000029	102025	Authentic Brands
9	C0000030	062025	Recurrent

Select * From Movies;



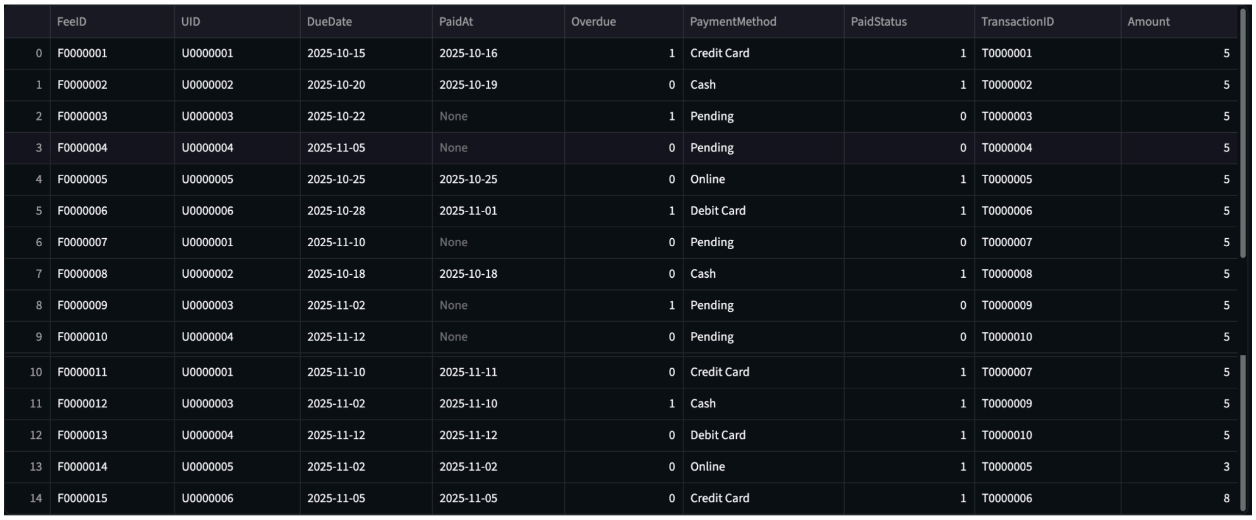
	CopyID	Rating	Director
0	C0000011	87%	Christopher Nolan
1	C0000012	92%	Wes Anderson
2	C0000013	94%	Denis Villeneuve
3	C0000014	96%	David Fincher
4	C0000015	97%	Bob Persichetti
5	C0000016	73%	Christopher Nolan
6	C0000017	91%	Damien Chazelle
7	C0000018	97%	George Miller
8	C0000019	91%	Ridley Scott
9	C0000020	98%	Pete Docter

Select * From Reserves;



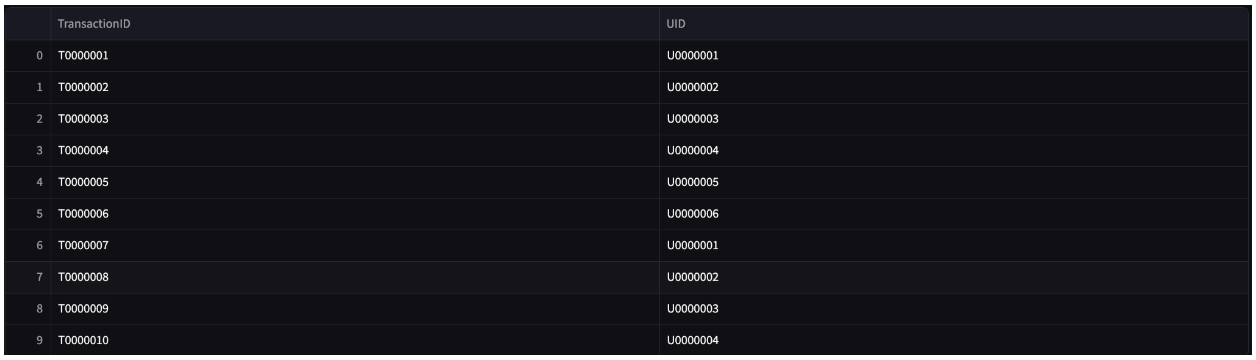
	CopyID	UID	ReservedAt
0	C0000001	U0000001	2025-09-15
1	C0000002	U0000002	2025-09-16
2	C0000003	U0000003	2025-09-17
3	C0000004	U0000004	2025-09-18
4	C0000005	U0000005	2025-09-19
5	C0000006	U0000006	2025-09-20
6	C0000007	U0000001	2025-09-21
7	C0000008	U0000002	2025-09-22
8	C0000009	U0000003	2025-09-23
9	C0000010	U0000004	2025-09-24

Select * From Fee;



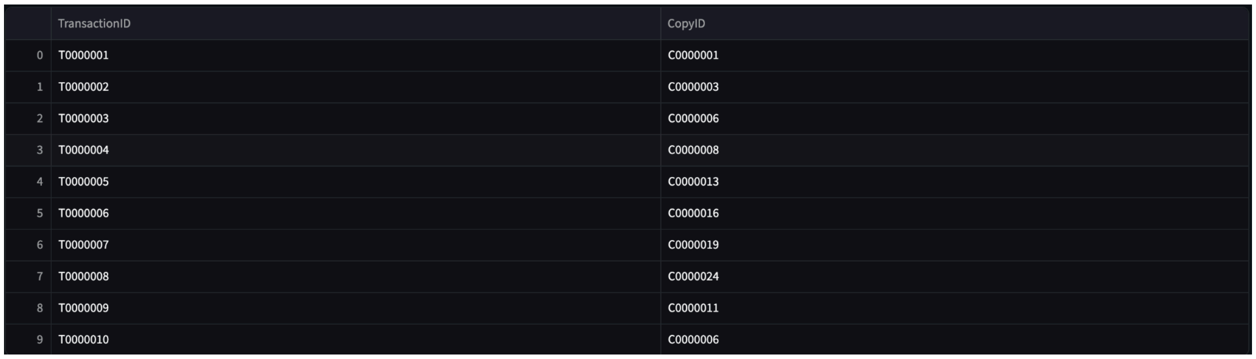
	FeeID	UID	DueDate	PaidAt	Overdue	PaymentMethod	PaidStatus	TransactionID	Amount
0	F0000001	U0000001	2025-10-15	2025-10-16	1	Credit Card	1	T0000001	5
1	F0000002	U0000002	2025-10-20	2025-10-19	0	Cash	1	T0000002	5
2	F0000003	U0000003	2025-10-22	None	1	Pending	0	T0000003	5
3	F0000004	U0000004	2025-11-05	None	0	Pending	0	T0000004	5
4	F0000005	U0000005	2025-10-25	2025-10-25	0	Online	1	T0000005	5
5	F0000006	U0000006	2025-10-28	2025-11-01	1	Debit Card	1	T0000006	5
6	F0000007	U0000001	2025-11-10	None	0	Pending	0	T0000007	5
7	F0000008	U0000002	2025-10-18	2025-10-18	0	Cash	1	T0000008	5
8	F0000009	U0000003	2025-11-02	None	1	Pending	0	T0000009	5
9	F0000010	U0000004	2025-11-12	None	0	Pending	0	T0000010	5
10	F0000011	U0000001	2025-11-10	2025-11-11	0	Credit Card	1	T0000007	5
11	F0000012	U0000003	2025-11-02	2025-11-10	1	Cash	1	T0000009	5
12	F0000013	U0000004	2025-11-12	2025-11-12	0	Debit Card	1	T0000010	5
13	F0000014	U0000005	2025-11-02	2025-11-02	0	Online	1	T0000005	3
14	F0000015	U0000006	2025-11-05	2025-11-05	0	Credit Card	1	T0000006	8

Select * From Rents;



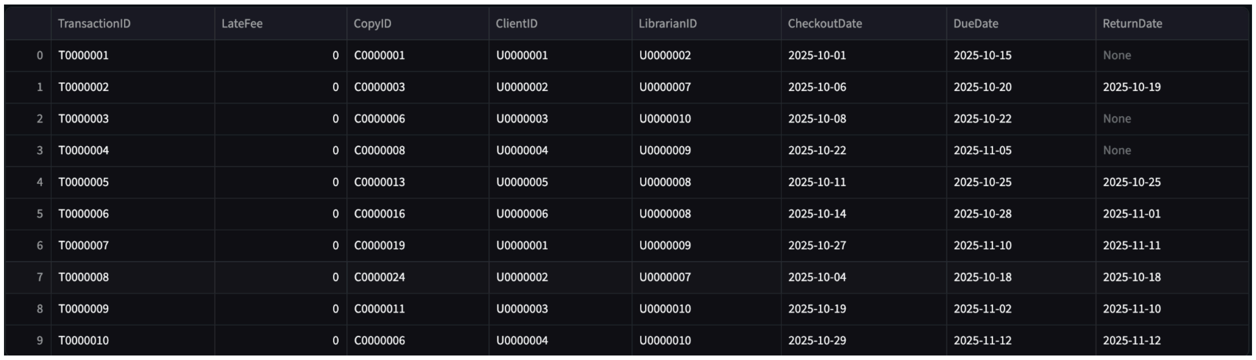
	TransactionID	UID
0	T0000001	U0000001
1	T0000002	U0000002
2	T0000003	U0000003
3	T0000004	U0000004
4	T0000005	U0000005
5	T0000006	U0000006
6	T0000007	U0000001
7	T0000008	U0000002
8	T0000009	U0000003
9	T0000010	U0000004

Select * From Rented;



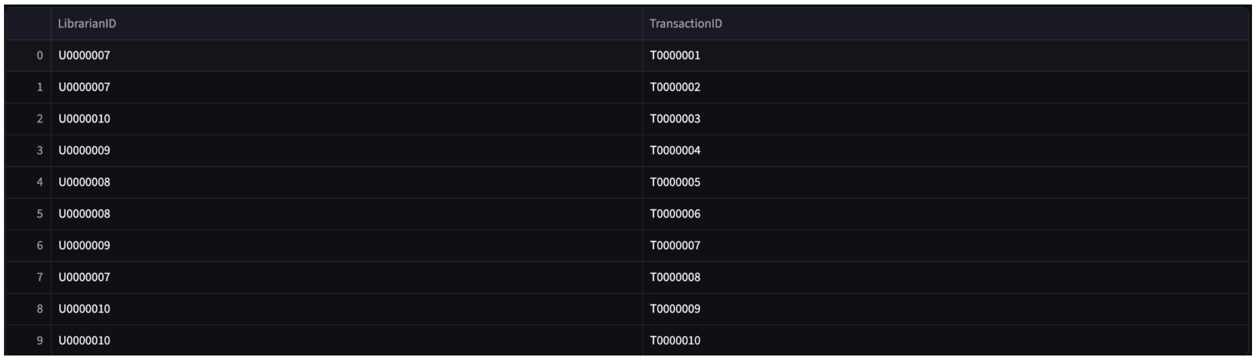
	TransactionID	CopyID
0	T0000001	C0000001
1	T0000002	C0000003
2	T0000003	C0000006
3	T0000004	C0000008
4	T0000005	C0000013
5	T0000006	C0000016
6	T0000007	C0000019
7	T0000008	C0000024
8	T0000009	C0000011
9	T0000010	C0000006

Select * From UserTransaction;



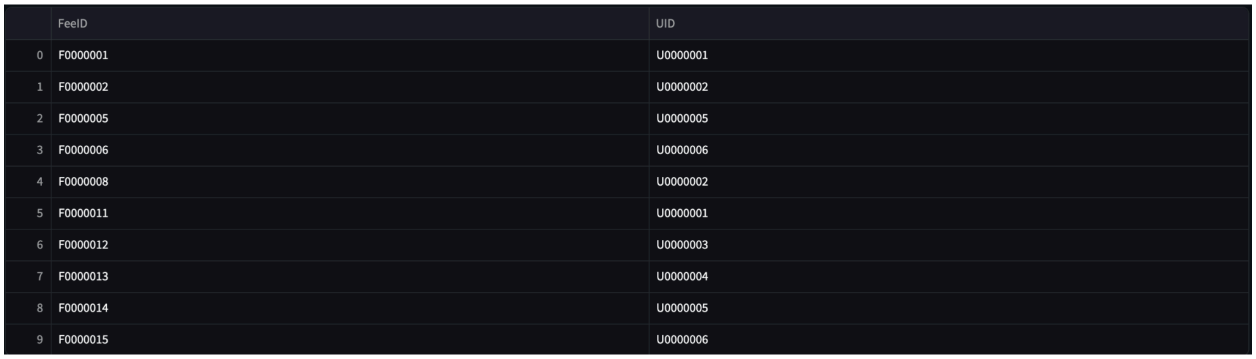
	TransactionID	LateFee	CopyID	ClientID	LibrarianID	CheckoutDate	DueDate	ReturnDate
0	T0000001	0	C0000001	U0000001	U0000002	2025-10-01	2025-10-15	None
1	T0000002	0	C0000003	U0000002	U0000007	2025-10-06	2025-10-20	2025-10-19
2	T0000003	0	C0000006	U0000003	U0000010	2025-10-08	2025-10-22	None
3	T0000004	0	C0000008	U0000004	U0000009	2025-10-22	2025-11-05	None
4	T0000005	0	C0000013	U0000005	U0000008	2025-10-11	2025-10-25	2025-10-25
5	T0000006	0	C0000016	U0000006	U0000008	2025-10-14	2025-10-28	2025-11-01
6	T0000007	0	C0000019	U0000001	U0000009	2025-10-27	2025-11-10	2025-11-11
7	T0000008	0	C0000024	U0000002	U0000007	2025-10-04	2025-10-18	2025-10-18
8	T0000009	0	C0000011	U0000003	U0000010	2025-10-19	2025-11-02	2025-11-10
9	T0000010	0	C0000006	U0000004	U0000010	2025-10-29	2025-11-12	2025-11-12

Select * From Assists;



	LibrarianID	TransactionID
0	U0000007	T0000001
1	U0000007	T0000002
2	U0000010	T0000003
3	U0000009	T0000004
4	U0000008	T0000005
5	U0000008	T0000006
6	U0000009	T0000007
7	U0000007	T0000008
8	U0000010	T0000009
9	U0000010	T0000010

Select * From Pays;



	FeeID	UID
0	F0000001	U0000001
1	F0000002	U0000002
2	F0000005	U0000005
3	F0000006	U0000006
4	F0000008	U0000002
5	F0000011	U0000001
6	F0000012	U0000003
7	F0000013	U0000004
8	F0000014	U0000005
9	F0000015	U0000006