GROUP 17 - Online Vinyl Record Store - Daria Solomon, Timeea Andreea Radu, Ioana Lupu

Project Description

The primary function of the vinyl record web application is to provide users with a smooth and user-friendly experience for browsing and purchasing vinyl records, while also offering comprehensive account management features. Through the platform, users can explore a vast catalogue of available vinyl records, each accompanied by detailed information such as the title, artist, genre, price, release date and current availability status. This allows users to easily discover new music or find specific records of interest.

In addition to browsing and buying records, the system supports different customer account types—Regular, Premium, and VIP. Each customer type comes with unique benefits tailored to their subscription tier. Premium and VIP customers enjoy additional perks based on the subscription fees they choose to pay, with Premium Fee and VIP Fee offering varying levels of access and advantages, such as discounts. Regular users may not have access to the same exclusive features as Premium and VIP customers, creating a tiered system that rewards higher levels of membership. The platform makes it easy for users to upgrade their membership, manage their preferences, and review past purchases, ensuring a seamless and personalised experience for every user.

Mapping Approach and Code Snippets:

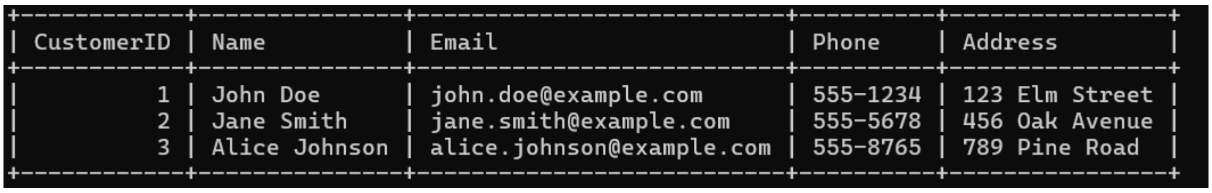
Our mapping approach involves creating tables to keep track of entity relationships.
In our project we are using three ISA relations in our Schemas.
We filled the tables with dummy data for a better visualisation and understanding.

- The first one, which categorises customers into Regular Customer, Premium Customer and VIP Customer, is a type 1 ISA relation. Therefore we have created 4 tables corresponding to the number of entities. The higher-level entity set is the Customer and the rest are low-level. The Regular Customer can't be a Premium Customer or a VIP Customer, and vice versa. Note that the low-level entities' keys are the primary key of Customers.

Relevant tables:

1. **Customer Table:**

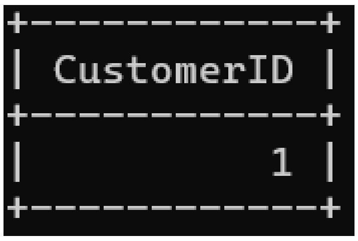
```
CREATE TABLE `Customer` (
  `CustomerID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` varchar(100) NOT NULL,
  `Email` varchar(100) NOT NULL,
  `Phone` varchar(20) DEFAULT NULL,
  `Address` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`CustomerID`),
  UNIQUE KEY `Email` (`Email`)
);
```



```
+------------+---------------+---------------------------+----------+----------------+
| CustomerID | Name          | Email                     | Phone    | Address        |
+------------+---------------+---------------------------+----------+----------------+
|          1 | John Doe      | john.doe@example.com      | 555-1234 | 123 Elm Street |
|          2 | Jane Smith    | jane.smith@example.com    | 555-5678 | 456 Oak Avenue |
|          3 | Alice Johnson | alice.johnson@example.com | 555-8765 | 789 Pine Road  |
+------------+---------------+---------------------------+----------+----------------+
```

2. **Regular Customer Table:**

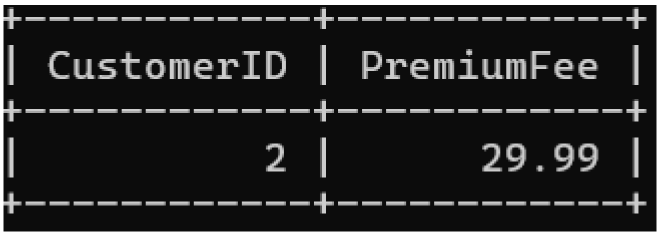
```
CREATE TABLE `RegularCustomer` (
  `CustomerID` int(11) NOT NULL,
  PRIMARY KEY (`CustomerID`),
  CONSTRAINT `RegularCustomer_ibfk_1` FOREIGN KEY (`CustomerID`) REFERENCES
`Customer` (`CustomerID`)
);
```



```
+------------+
| CustomerID |
+------------+
|          1 |
+------------+
```

3. **Premium Customer Table:**

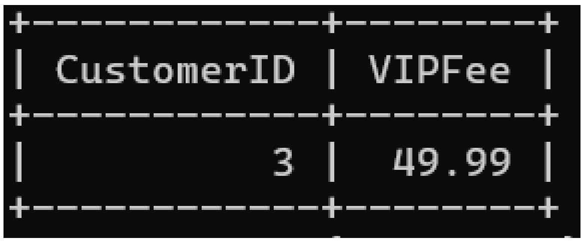
```
 CREATE TABLE `PremiumCustomer` (
  `CustomerID` int(11) NOT NULL,
  `PremiumFee` decimal(10,2) NOT NULL,
  PRIMARY KEY (`CustomerID`),
  CONSTRAINT `PremiumCustomer_ibfk_1` FOREIGN KEY (`CustomerID`) REFERENCES
`Customer` (`CustomerID`)
);
```



```
+------------+------------+
| CustomerID | PremiumFee |
+------------+------------+
|          2 |      29.99 |
+------------+------------+
```

4. **VIP Customer Table:**

```
 CREATE TABLE `VIPCustomer` (
  `CustomerID` int(11) NOT NULL,
  `VIPFee` decimal(10,2) NOT NULL,
  PRIMARY KEY (`CustomerID`),
  CONSTRAINT `VIPCustomer_ibfk_1` FOREIGN KEY (`CustomerID`) REFERENCES
`Customer` (`CustomerID`)
);
```



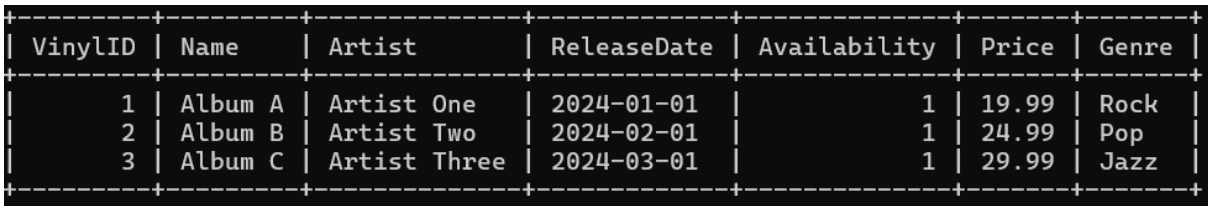
```
+------------+--------+
| CustomerID | VIPFee |
+------------+--------+
|          3 |  49.99 |
+------------+--------+
```

- The second ISA Relation is composed of a high level entity set, Vinyl, and 3 lower-level ones, LP, EP and Single. Thus, we have another Type 1 ISA relation. The vinyl can be only of one type LP, EP, or Single.

Relevant tables:

1. **Vinyl Table:**

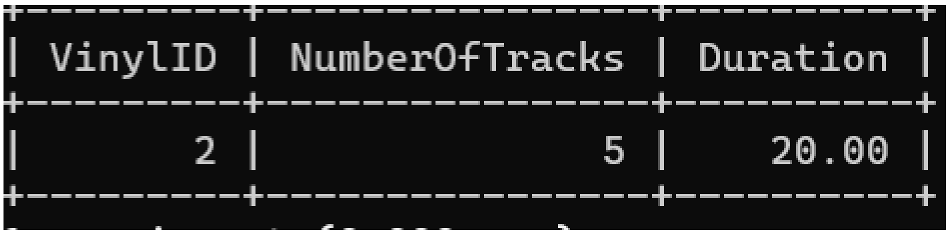
```
CREATE TABLE `Vinyl` (
  `VinylID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` varchar(100) NOT NULL,
  `Artist` varchar(100) NOT NULL,
  `ReleaseDate` date DEFAULT NULL,
  `Availability` tinyint(1) DEFAULT 1,
  `Price` decimal(10,2) NOT NULL,
  `Genre` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`VinylID`)
);
```



```
+---------+---------+--------------+-------------+--------------+-------+-------+
| VinylID | Name    | Artist       | ReleaseDate | Availability | Price | Genre |
+---------+---------+--------------+-------------+--------------+-------+-------+
|       1 | Album A | Artist One   | 2024-01-01  |            1 | 19.99 | Rock  |
|       2 | Album B | Artist Two   | 2024-02-01  |            1 | 24.99 | Pop   |
|       3 | Album C | Artist Three | 2024-03-01  |            1 | 29.99 | Jazz  |
+---------+---------+--------------+-------------+--------------+-------+-------+
```

2. **Extended Play Table(EP):**

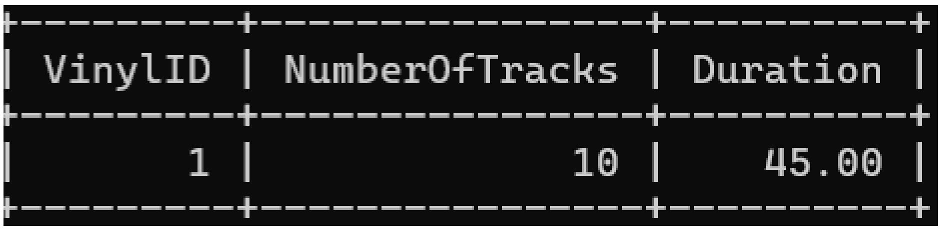
```
 CREATE TABLE `EP` (
  `VinylID` int(11) NOT NULL,
  `NumberOfTracks` int(11) DEFAULT NULL,
  `Duration` decimal(5,2) DEFAULT NULL,
  PRIMARY KEY (`VinylID`),
  CONSTRAINT `EP_ibfk_1` FOREIGN KEY (`VinylID`) REFERENCES `Vinyl` (`VinylID`)
);
```



```
+---------+----------------+----------+
| VinylID | NumberOfTracks | Duration |
+---------+----------------+----------+
|       2 |              5 |    20.00 |
+---------+----------------+----------+
```

3. **Long Play Table(LP):**

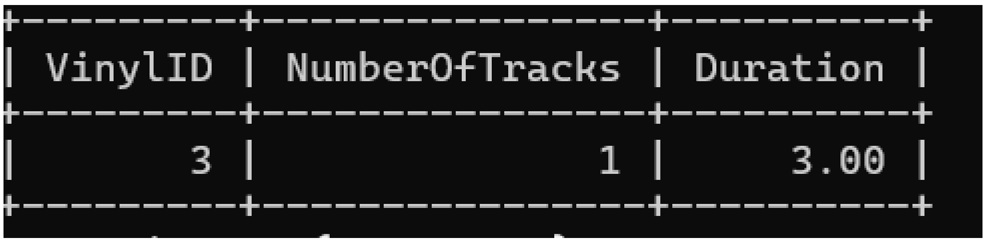
```
CREATE TABLE `LP` (
  `VinylID` int(11) NOT NULL,
  `NumberOfTracks` int(11) DEFAULT NULL,
  `Duration` decimal(5,2) DEFAULT NULL,
  PRIMARY KEY (`VinylID`),
  CONSTRAINT `LP_ibfk_1` FOREIGN KEY (`VinylID`) REFERENCES `Vinyl` (`VinylID`)
);
```



```
+---------+----------------+----------+
| VinylID | NumberOfTracks | Duration |
+---------+----------------+----------+
|       1 |             10 |    45.00 |
+---------+----------------+----------+
```

4. **Single Table:**

```
CREATE TABLE `Single` (
  `VinylID` int(11) NOT NULL,
  `NumberOfTracks` int(11) DEFAULT NULL,
  `Duration` decimal(5,2) DEFAULT NULL,
  PRIMARY KEY (`VinylID`),
  CONSTRAINT `Single_ibfk_1` FOREIGN KEY (`VinylID`) REFERENCES `Vinyl` (`VinylID`)
);
```



```
+---------+----------------+----------+
| VinylID | NumberOfTracks | Duration |
+---------+----------------+----------+
|       3 |              1 |     3.00 |
+---------+----------------+----------+
```

- The Fee table represents a Type 3 mapping (single table for all subclasses and superclass). In this case, all fee types (Regular, Premium, VIP) are stored in a single table, and the FeeType column acts as a discriminator to differentiate between the various subclasses.

- In the Payments table, we also used a Type 3 (single relation) ISA mapping approach to simplify the handling of various customer transactions, including membership fees and vinyl purchases. This method consolidates all payment-related information—such as customer ID, vinyl ID, discounts, and final amounts—into a single table, making queries and data management more straightforward.Instead of splitting the table into separate tables for different payment types (like "Vinyl Purchase" or "Membership Fee"), you use a single table and differentiate between types using the PaymentFor column, which is an enum.

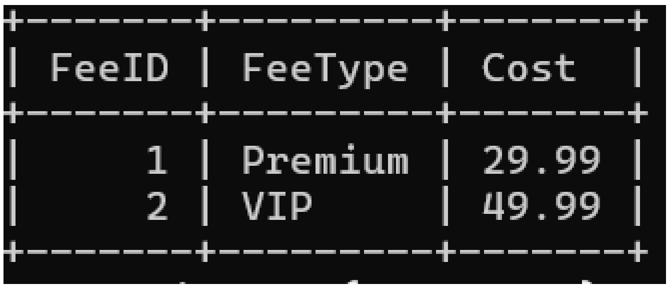
- This way, some values remain unused for fields unrelated to certain purchases. For example, no discount is applied if the payment is done by a RegularCustomer or if someone paid for a membership fee- so the Applied Discount and Discount ID fields are NULL. Alternatively, we could have used a Type 1 mapping, where each entity (e.g., Regular, Premium, and VIP customers) would have its own separate table for their purchases..
  We chose Type 3 to avoid complex joins and to keep data centralised, while still capturing key relationships such as customer types and their respective discounts. This approach simplifies getting payment data and maintains referential integrity through foreign keys.

- The Discounts table would be used to apply discounts during transactions. When processing payments or purchases, the Discounts table helps determine the applicable discount based on the customer type and calculates the final amount accordingly, which is shown in the Payments table.

Relevant tables:

1. **Fee Table:**

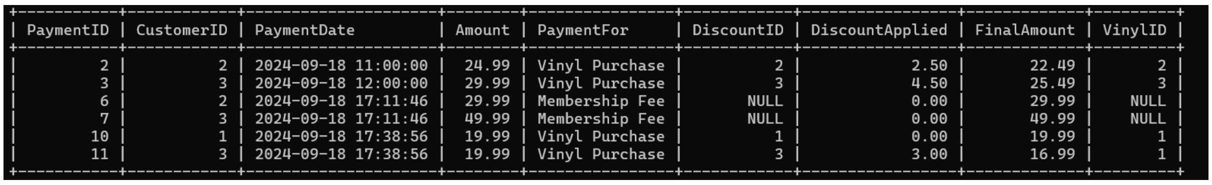
```
CREATE TABLE `Fee` (
  `FeeID` int(11) NOT NULL AUTO_INCREMENT,
  `FeeType` enum('Regular','Premium','VIP') DEFAULT NULL,
  `Cost` decimal(10,2) NOT NULL,
  PRIMARY KEY (`FeeID`)
);
```



```
+-------+---------+-------+
| FeeID | FeeType | Cost  |
+-------+---------+-------+
|     1 | Premium | 29.99 |
|     2 | VIP     | 49.99 |
+-------+---------+-------+
```

2. **Payments Table:**

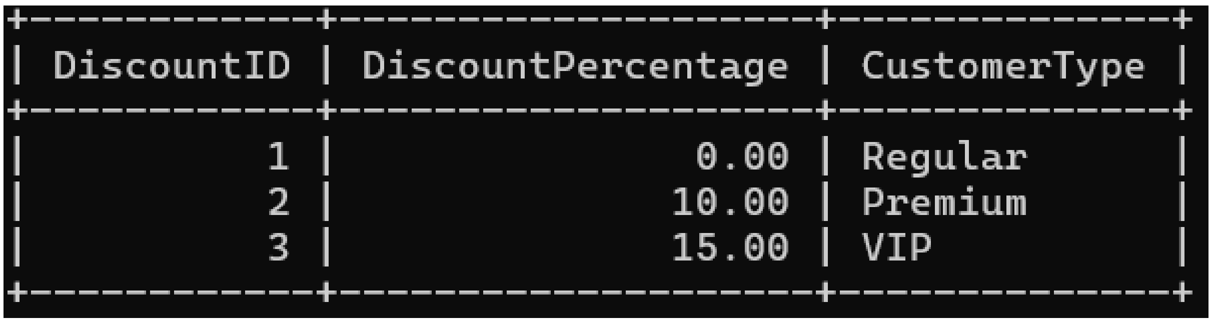
```
 CREATE TABLE `Payments` (
  `PaymentID` int(11) NOT NULL AUTO_INCREMENT,
  `CustomerID` int(11) DEFAULT NULL,
  `PaymentDate` timestamp NOT NULL DEFAULT current_timestamp(),
  `Amount` decimal(10,2) NOT NULL,
  `PaymentFor` enum('Membership Fee','Vinyl Purchase','Other') DEFAULT 'Other',
  `DiscountID` int(11) DEFAULT NULL,
  `DiscountApplied` decimal(10,2) DEFAULT 0.00,
  `FinalAmount` decimal(10,2) DEFAULT 0.00,
  `VinylID` int(11) DEFAULT NULL,
  PRIMARY KEY (`PaymentID`),
  KEY `CustomerID` (`CustomerID`),
  KEY `DiscountID` (`DiscountID`),
  CONSTRAINT `Payments_ibfk_1` FOREIGN KEY (`CustomerID`) REFERENCES
`Customer` (`CustomerID`),
  CONSTRAINT `Payments_ibfk_2` FOREIGN KEY (`DiscountID`) REFERENCES
`Discounts` (`DiscountID`)
);
```



```
+-----------+------------+---------------------+--------+----------------+------------+-----------------+-------------+---------+
| PaymentID | CustomerID | PaymentDate         | Amount | PaymentFor     | DiscountID | DiscountApplied | FinalAmount | VinylID |
+-----------+------------+---------------------+--------+----------------+------------+-----------------+-------------+---------+
|         2 |          2 | 2024-09-18 11:00:00 |  24.99 | Vinyl Purchase |          2 |            2.50 |       22.49 |       2 |
|         3 |          3 | 2024-09-18 12:00:00 |  29.99 | Vinyl Purchase |          3 |            4.50 |       25.49 |       3 |
|         6 |          2 | 2024-09-18 17:11:46 |  29.99 | Membership Fee |       NULL |            0.00 |       29.99 |    NULL |
|         7 |          3 | 2024-09-18 17:11:46 |  49.99 | Membership Fee |       NULL |            0.00 |       49.99 |    NULL |
|        10 |          1 | 2024-09-18 17:38:56 |  19.99 | Vinyl Purchase |          1 |            0.00 |       19.99 |       1 |
|        11 |          3 | 2024-09-18 17:38:56 |  19.99 | Vinyl Purchase |          3 |            3.00 |       16.99 |       1 |
+-----------+------------+---------------------+--------+----------------+------------+-----------------+-------------+---------+
```

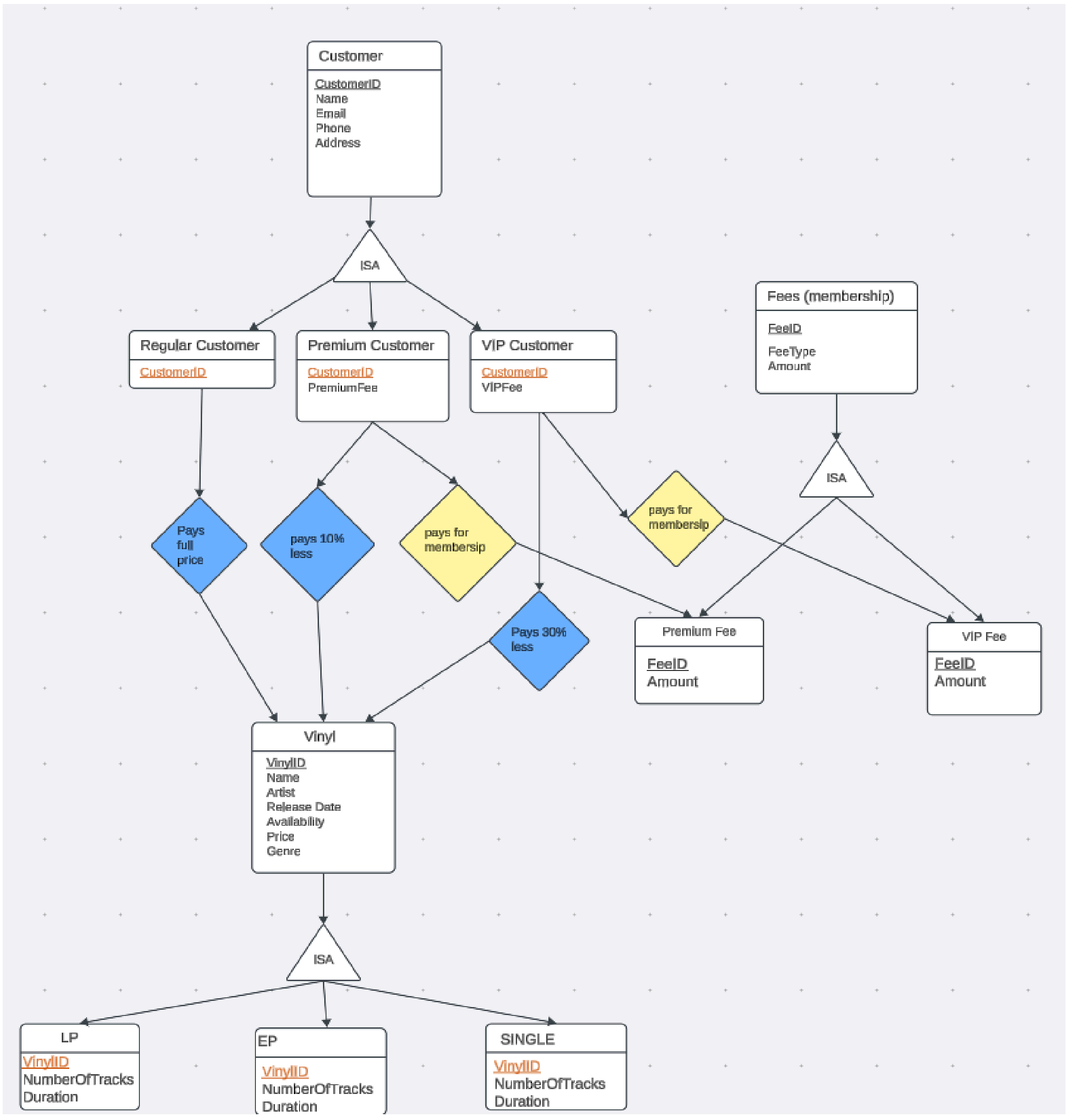
3. **Discounts Table:**

```
 CREATE TABLE `Discounts` (
  `DiscountID` int(11) NOT NULL AUTO_INCREMENT,
  `DiscountPercentage` decimal(5,2) NOT NULL,
  `CustomerType` enum('Regular','Premium','VIP') NOT NULL,
  PRIMARY KEY (`DiscountID`)
);
```



```
+------------+--------------------+--------------+
| DiscountID | DiscountPercentage | CustomerType |
+------------+--------------------+--------------+
|          1 |               0.00 | Regular      |
|          2 |              10.00 | Premium      |
|          3 |              15.00 | VIP          |
+------------+--------------------+--------------+
```

Updated ER Diagram



NOTES:

- In this diagram you can see that foreign keys are orange.

FeeID is not orange because in our SQL schema we decided to record all of these fee types in one table.

- In our schema, we represent the relationships with specific tables: the Discounts table for discounts and the Paymentstable for recording all payments. The reason for this setup is that the interaction between a customer's membership type and their purchase of vinyls impacts the payment process. Membership type influences whether a discount is applied to vinyl purchases, and it also affects whether a membership fee has been paid.

- All payments are recorded in the Payments table. If the payment is for a membership fee, we record the full membership fee based on the customer's subclass, such as Regular, Premium, or VIP. In this case, no discount is applied. The Payments table traces each CustomerID to their specific subclass, and the membership fee is retrieved from the relevant subclass table.
- For vinyl purchases, the process is different. We check the customer's type and apply the appropriate discount by referencing the Discounts table. The Payments table records the FinalAmount and calculates the discount amount based on this ID. The final payment amount is computed after applying the discount, and this amount is recorded as the Final Amount in the Payments table.

We felt the need to clarify what may come across as an unconventional approach to translating entity relationships into SQL schemas.