NAME: LAMIN CHATTY

MAT #: 286905

DATABASE MANAGEMENT SYSTEM PROJECT [DT0347]

Requirements

R1: The Database will store customer registration details to the services such as (full name, address, e-mail, user_id).

R2: The service will provide a wide range of audio content, including music albums and singles from various music production companies, a diverse selection of podcasts, an extensive collection of audiobooks, and access to radio stations, live-streaming capabilities, and a platform for users to share and enjoy their own audio tracks.

R3: Customer can listen to free content offered by the service.

R4: For each content offered by the service it will include detail such as content_ title, content_ type, price, release date and access type. Also it will differentiate between content owned by companies and user-created content.

R5: The owner of a content can set price to access their individual contents.

R6: The Database will include Subscription and the subscription will include customer details, start date, end date, subscription plan, Subscription_id and type.

R7: Customers can create Playlist which can be made private or public.

Use Cases

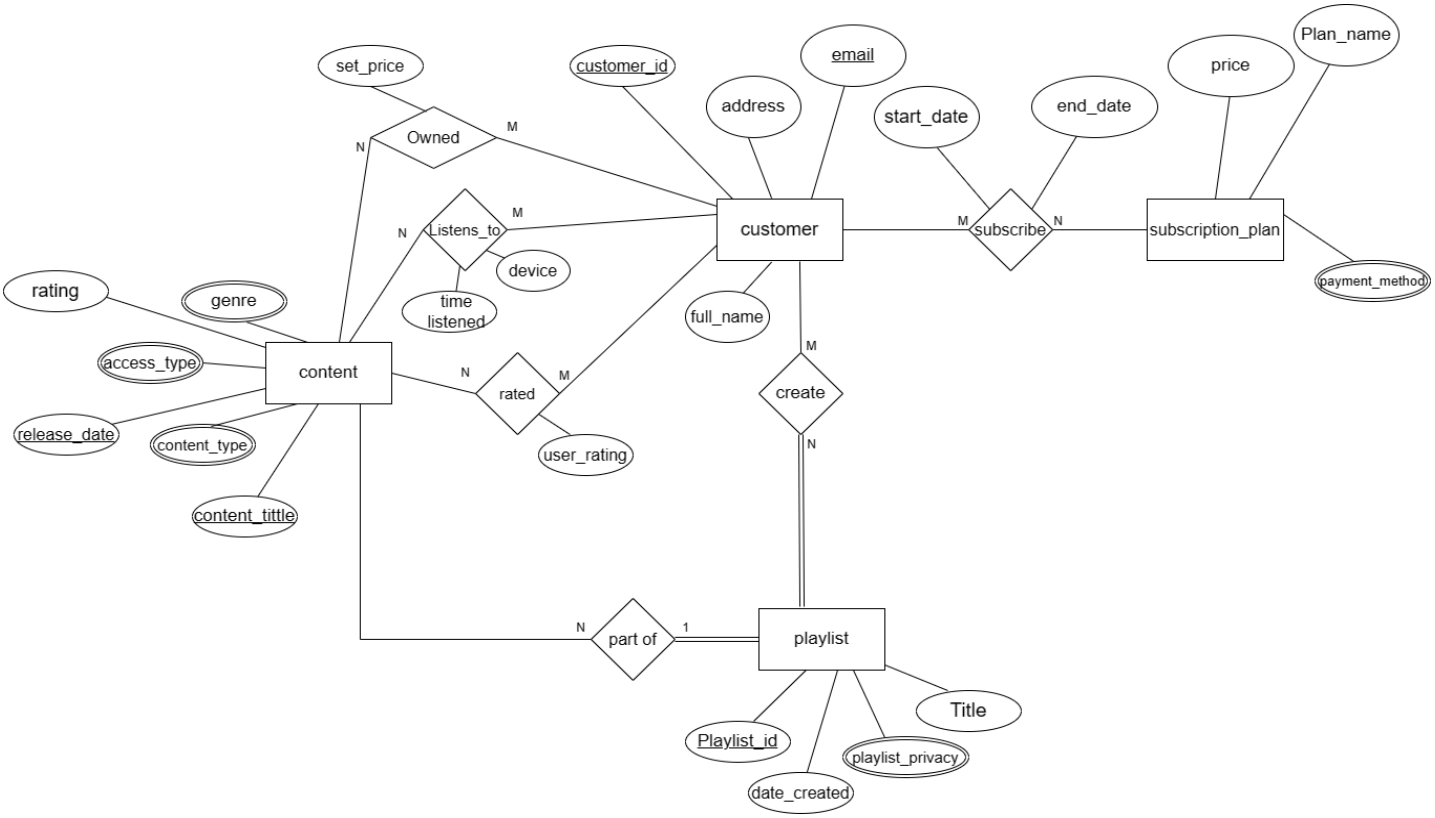
U1: The platform enables personalized content exploration based on user preferences, including type and genre as well as playlist.

U2: The service administrators have the authority to generate a list of Customer_id for customers who have not renewed their subscriptions. This list also provides details about the types of subscriptions these users previously had and the dateline for the expiry of their subscription can be communicated to them.

U3: Users can list the audio contents of a given type by setting a minimum rating on a scale of 1 to 5.

U4: Based on user profiles, the service offers personalized content recommendations to enhance user experience.

Conceptual design



Entity type

- Customer [R1,R3,R7,R5,U1,U3]
- Content [R3,R4,R5, U4,U5]
- Subscription [R6, U2]
- Playlist[R7, U1]

Relationship types:

- Customer (M)<Subscribe> Subscription (N) [R6,U2]
- Customer (M) < Create > Playlist(N)[R7]
- Customer (M)< Rates > Content (N)[U3]
- Playlist (1)< Part of>Content (N)
- Customer(M)<Listen_to> Content(N)
- Customer(M)<Owned> Content(N)[R5]

Logical design

STEP 1: Mapping of Regular Strong Entity Types

Each regular (strong) entity type which includes customer, content, playlist and subscription should have a corresponding relation R that includes all the simple attributes of the entity. The primary key for is selected from one of the key attributes of entity, and if the chosen key is composite, the set of simple attributes forming it will serve as the primary key.

Customer

Customer_id	Address	Full name	Email

Content

Content_title	Release_date	Ratings

Playlist

Title	Playlist_id	date created

Subscription Plan

Plan_name	Price

Step 2: Mapping of Weak Entity Types

The Database does not include any weak entity type.

Step 3: Mapping of Binary 1:1 Relation Types

The Database does not include any one to one relationship type

STEP 3: Mapping of Binary 1: N Relationship type

To represent the participation of the N-side entity type in a regular binary 1: N relationship type, identify the relation that corresponds to the participating entity type. In the Content relation, I include the primary key from the Playlist, which represent the other participating entity types in the "Part of" relationship, as foreign keys.

Content

Content_title	Release_date	Playlist_id	Rating

Step 3: To represent a regular binary M: N relationship type R, a new relation S is created as a relationship relation. The primary keys of the participating entity types' relations are included as foreign key attributes in S, and their combination forms the primary key of S. Additionally, any simple attributes from the original M: N relationship types (or simple components of composite attributes) are included as attributes in S.

Owned

Content_title	Customer_id	Set price	Release_date

Rated

Content_title	Release_date	Customer_id	User_rating

Listen_to

Customer_id	Content_title	Release_date	Time Listened	Device used

Subscribe

Plan_name	Customer_id	Start_date	End_date

Create

Customer_id	Playlist_title	Playlist_id

STEP 4: Mapping of Multivalued attributes.

For each multivalued attribute of relation R, create two new relations: A and K. The relation K is responsible for holding the values of the multivalued attribute. The relation A is responsible for maintaining the relationships between R and K. Relation A includes a foreign key attribute corresponding to the primary key of R, along with a foreign key attribute representing the primary key of K. The primary key of A is formed by the combination of the primary keys of R and K. If the multivalued attribute is composite, the simple attributes that compose it are included in relation K.

Content_type

content_type_name

Content _to_type

Content_title	Release_date	Content_type_name

Content_genre

Genre_name

Content_to_genre

Content_title	Release_date	Genre_name

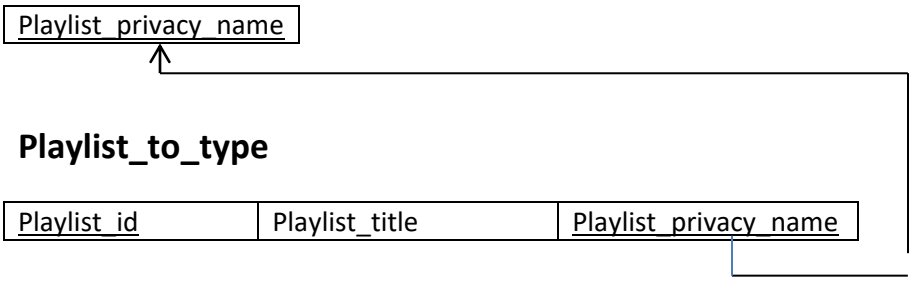
Access_type

Access_type_name

Content_to_access

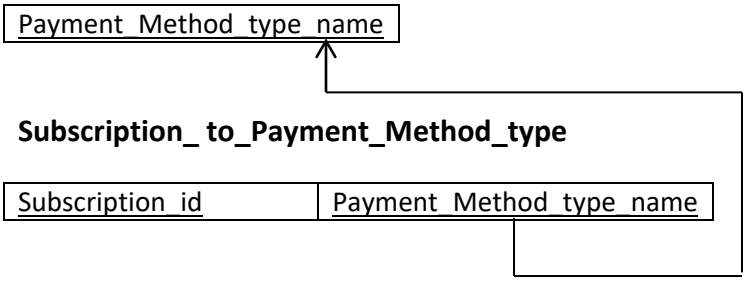
Content_title	Release_date	Access_type_name

Playlist_ privacy



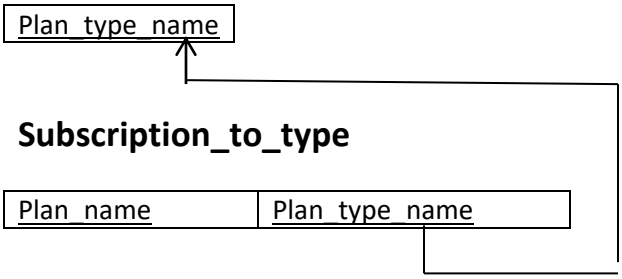
Playlist_to_type

Payment Method_type



Subscription_ to_Payment_Method_type

Plan_type



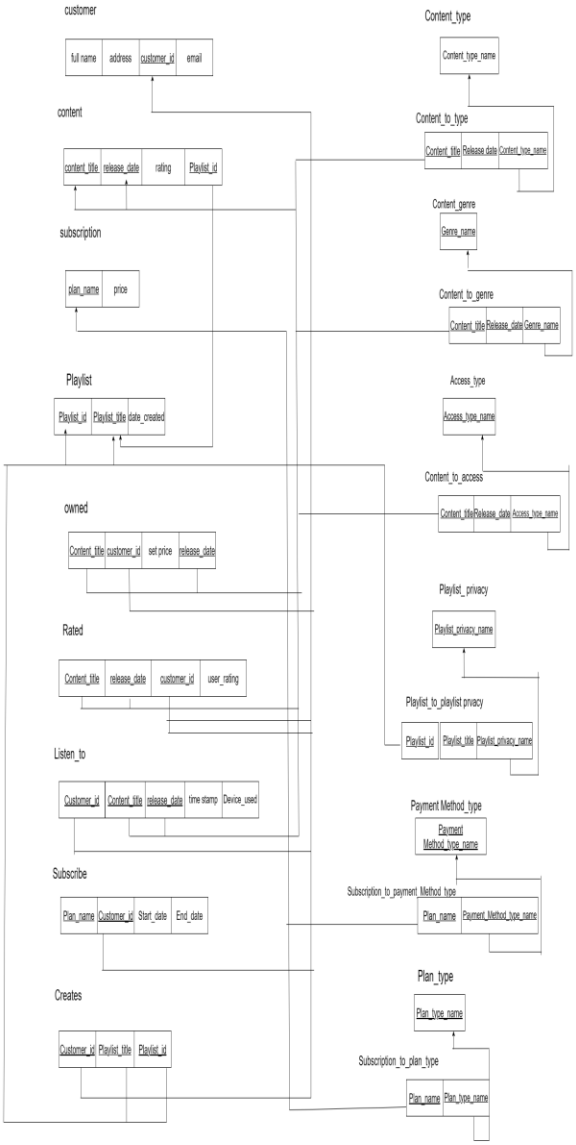
Subscription_to_type

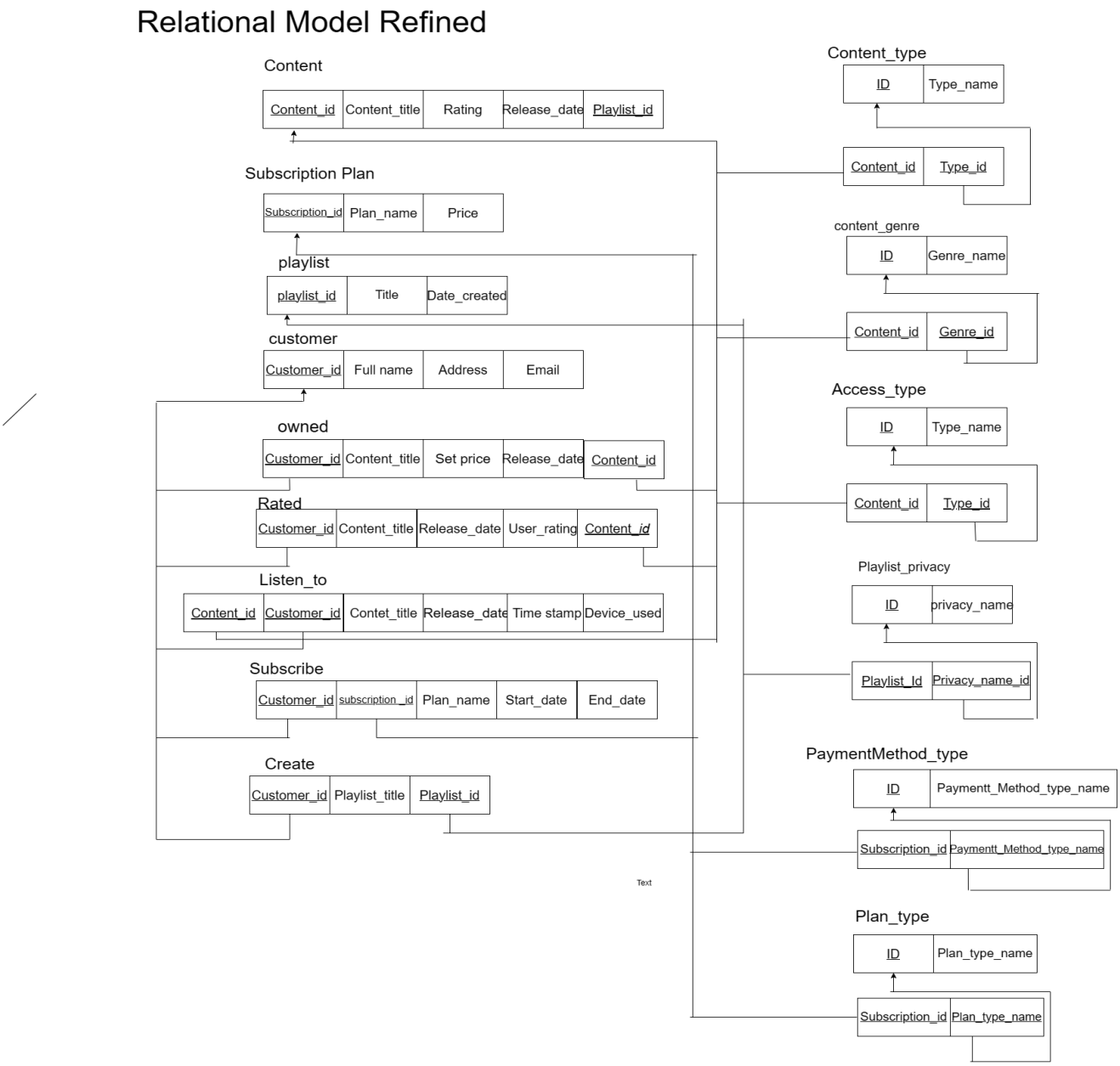
Step 7: Mapping of N-ary Relationship Types.

The Database does not have any n-ary relationship type.

Relational model

Relational Model





SQL schema implementation

CREATE TABLE Customer (**customer_id INT PRIMARY KEY auto_increment NOT NULL,** **full_name VARCHAR(100) NOT NULL,** **address VARCHAR(200) NOT NULL,** **email VARCHAR(100) NOT NULL** **);**	**CREATE TABLE rated (** **user_rating INT NOT NULL,** **release_data VARCHAR(255) NOT NULL,** **content_id INT NOT NULL,** **customer_id INT NOT NULL,** **CHECK (user_rating >= 1 AND user_rating <= 5),** **FOREIGN KEY (content_id) REFERENCES content(content_id),**

	FOREIGN KEY (customer_id) REFERENCES customer(customer_id), PRIMARY KEY (content_id, customer_id));
CREATE TABLE SubscriptionPlan (subscription_id INT PRIMARY KEY auto_increment NOT NULL, plan_name VARCHAR(255) NOT NULL, price INT NOT NULL);	CREATE TABLE Owned (set_price INT NOT NULL, customer_id INT NOT NULL, content_id INT NOT NULL, content_title VARCHAR(100) NOT NULL, release_date DATE NOT NULL, PRIMARY KEY (content_id, customer_id), FOREIGN KEY (customer_id) REFERENCES Customer(customer_id), FOREIGN KEY (content_id) REFERENCES Content(content_id));
CREATE TABLE playlist (Playlist_id INT PRIMARY KEY NOT NULL, title VARCHAR(255) NOT NULL, date_created DATE NOT NULL);	CREATE TABLE Listen_to (customer_id INT, content_id INT, content_title VARCHAR(100), release_date DATE, time_listened time NOT NULL, device_use VARCHAR(50), PRIMARY KEY (customer_id, content_id), FOREIGN KEY (customer_id) REFERENCES Customer(customer_id), FOREIGN KEY (content_id) REFERENCES Content(content_id));
CREATE TABLE Content (content_id INT PRIMARY KEY, content_title VARCHAR(100), rating INT CHECK (rating >= 1 AND rating <= 5), release_date DATE, playlist_id INT, FOREIGN KEY (playlist_id) REFERENCES Playlist(playlist_id));	CREATE TABLE Subscribe (customer_id INT NOT NULL auto_increment, subscription_id INT NOT NULL, plan_name VARCHAR(100) NOT NULL, start_date DATE NOT NULL, end_date DATE NOT NULL, PRIMARY KEY (customer_id, subscription_id), FOREIGN KEY (customer_id) REFERENCES Customer(customer_id), FOREIGN KEY (subscription_id) REFERENCES SubscriptionPlan(subscription_id));
CREATE TABLE createe (customer_id INT NOT NULL auto_increment,	CREATE TABLE content_type (id INT NOT NULL AUTO_INCREMENT, Type_name VARCHAR(255) NOT NULL,

playlist_id INT PRIMARY KEY NOT NULL, playlist_title VARCHAR(100), FOREIGN KEY (customer_id) REFERENCES Customer(customer_id));	PRIMARY KEY (id), UNIQUE (Type_name));
CREATE TABLE content_to_type (content_id INT NOT NULL, type_id INT NOT NULL, FOREIGN KEY (content_id) REFERENCES content(content_id), FOREIGN KEY (type_id) REFERENCES content_type(id));	CREATE TABLE content_genre (id INT NOT NULL AUTO_INCREMENT, Genre_name VARCHAR(255) NOT NULL, PRIMARY KEY (id), UNIQUE (Genre_name));
CREATE TABLE content_to_genre (content_id INT NOT NULL, genre_id INT NOT NULL, FOREIGN KEY (content_id) REFERENCES content(content_id), FOREIGN KEY (genre_id) REFERENCES content_genre(id));	CREATE TABLE Access_type (id INT NOT NULL AUTO_INCREMENT, Type_name VARCHAR(255) NOT NULL, PRIMARY KEY (id), UNIQUE (Type_name));
CREATE TABLE content_to_Access_type (content_id INT NOT NULL, Type_id INT NOT NULL, FOREIGN KEY (content_id) REFERENCES content(content_id), FOREIGN KEY (Type_id) REFERENCES Access_type(id));	CREATE TABLE Playlist_privacy(id INT NOT NULL AUTO_INCREMENT, privacy_name VARCHAR(255) NOT NULL, PRIMARY KEY (id), UNIQUE (privacy_name));
CREATE TABLE Playlist_to_playlist_privacy (playlist_id INT NOT NULL, privacy_name_id INT NOT NULL, FOREIGN KEY(playlist_id) REFERENCES playlist(playlist_id), FOREIGN KEY (privacy_name_id) REFERENCES Playlist_privacy(id));	CREATE TABLE PaymentMethod_type(id INT NOT NULL AUTO_INCREMENT, Paymentt_Method_type_name VARCHAR(255) NOT NULL, PRIMARY KEY (id), UNIQUE (Paymentt_Method_type_name));
CREATE TABLE Subscription_to_payment_Method_type (Subscription_id INT NOT NULL, Paymentt_Method_type_name_id INT NOT NULL, FOREIGN KEY(Subscription_id) REFERENCES subscriptionplan(Subscription_id), FOREIGN KEY	CREATE TABLE Plan_type(id INT NOT NULL AUTO_INCREMENT, Plan_type_name VARCHAR(255) NOT NULL, PRIMARY KEY (id), UNIQUE (Plan_type_name));

(Paymentt_Method_type_name_id) REFERENCES PaymentMethod_type(id));	
CREATE TABLE Subscription_to_plan_type(Subscription_id INT NOT NULL, Plan_type_name_id INT NOT NULL, FOREIGN KEY(Subscription_id) REFERENCES subscriptionplan(Subscription_id) FOREIGN KEY (Plan_type_name_id) REFERENCES Plan_type(id));	

Use cases implementation

1. (Use Case 1) the platform enables personalized content exploration based on user preferences, including type and genre.

```
SELECT c.*

FROM content AS c

INNER JOIN content_to_genre AS cg ON cg.content_id = c.content_id

INNER JOIN content_genre AS co ON co.id = id

WHERE co.Genre_name = 'AMPIANO'

AND release_date >= DATE_SUB(CURDATE(), INTERVAL 4 MONTH)

ORDER BY release_date DESC;
```

Output

The SQL code provided retrieves content records from the "content" table that belong to the genre 'AMPIANO', have a release date within the last 4 months, and sorts the results by release date in descending order.

- `SELECT c.*`: This specifies that we want to select all columns (`*`) from the "content" table. The results will include all columns of the "content" table for the matching records.

- `FROM content AS c`: It defines the table "content" as the source table for the query and assigns the alias "c" to it. The alias "c" can be used to refer to the "content" table in the rest of the query.

- `INNER JOIN content_to_genre AS cg ON cg.content_id = c.content_id`: This performs an inner join between the "content" and "content_to_genre" tables using the "content_id" column as

the join condition. It ensures that only records with matching content IDs from both tables are included in the result.

- `INNER JOIN content_genre AS co ON co.id = id`: This performs an inner join between the "content_to_genre" and "content_genre" tables using the "id" column as the join condition. It links the genre information with the content-to-genre mapping table.

- `WHERE co.Genre_name = 'AMPIANO' AND release_date >= DATE_SUB (CURDATE (), INTERVAL 4 MONTH)`: This sets the conditions for filtering the results. It specifies that only records with the genre name 'AMPIANO' and a release date within the last 4 months should be included.

- `ORDER BY release_date DESC`: This specifies that the resulting records should be sorted based on the release date column in descending order. The most recent releases will appear first in the result set.

2. (Use Case 2) the service administrators have the authority to generate a list of Customer_id for customers who have not renewed their subscriptions. This list also provides details about the types of subscriptions these users previously had and the dateline for the expiry of their subscription can be communicated to them.

SELECT subscribe.customer_id, subscribe.plan_name, subscribe.subscription_id, subscribe.end_date

FROM subscribe

WHERE subscribe.start_date < subscribe.end_date

GROUP BY subscribe.customer_id, subscribe.plan_name, subscribe.subscription_id, subscribe.end_date;

Output

- Output columns: The query selects specific columns (`customer_id`, `plan_name`, `subscription_id`, and `end_date`) from the "subscribe" table. These columns represent customer-related information such as their ID, subscription plan name, subscription ID, and the end date of the subscription.
- Filtering condition: The query applies a filtering condition (`subscribe.start_date < subscribe.end_date`) in the `WHERE` clause. This condition ensures that only active or valid subscriptions are included in the output by selecting rows where the start date of the subscription is earlier than the end date.
- Grouping: The query groups the result based on the columns `customer_id`, `plan_name`, `subscription_id`, and `end_date` using the `GROUP BY` clause. This grouping ensures that rows with the same combination of values in these columns are combined together in the output, providing a consolidated view of the distinct subscription records that satisfy the filtering condition.

3. (Use Case 3) Users can list the audio contents of a given type by setting a minimum rating on a scale of 1 to 5.

```
SELECT c.content_id, c.content_title, c.release_date, r.user_rating

FROM content c

JOIN rated r ON c.content_id = r.content_id

JOIN content_to_type ct ON c.content_id = ct.content_id

JOIN content_type t ON ct.type_id = t.id

WHERE t.Type_name = 'Music'

AND r.user_rating >= 3;
```

Output

- The query retrieves specific columns (`content_id`, `content_title`, `release_date`, `user_rating`) from multiple tables using joins and filters.
- Here's a summary of the expected outcome:
- Output columns: The result will include the unique identifier (`content_id`), title (`content_title`), release date (`release_date`), and user rating (`user_rating`) of content items.
- Joining tables: The query joins the `content`, `rated`, `content_to_type`, and `content_type` tables to establish connections between content, ratings, and content types.
- Filtering condition: The results are filtered to include only content items of type "Music" (`t.Type_name = 'Music'`) with a user rating of 3 or higher (`r.user_rating >= 3`).
- The outcome will be a collection of rows representing content items that meet the specified conditions, providing their ID, title, release date, and user rating.

4. (Use Case 4) based on user profiles, the service offers personalized content recommendations to enhance user experience.

```
SELECT c.content_id, c.content_title

FROM content c

WHERE c.release_date <= DATE_SUB (CURRENT_DAT E (), INTERVAL 30 DAY)

 AND c.content_id NOT IN (

 SELECT lt.content_id

  FROM listen_to lt

   WHERE lt.customer_id = 04

 )
```

ORDER BY RAND ()

LIMIT 5;

Output

The given SQL query retrieves specific columns from the "content" table based on certain conditions and ordering.

1. Output columns:

- `content_id`: This column represents the unique identifier of a content item.
- `content_title`: This column contains the title or name of the content item.

2. Filtering condition:

- The query filters the results based on two conditions:
  - `c.release_date <= DATE_SUB (CURRENT_DATE(), INTERVAL 30 DAY)`: This condition selects content items whose release date is within the last 30 days from the current date. In other words, it retrieves recently released content.

  - `c.content_id NOT IN (SELECT lt.content_id FROM listen_to lt WHERE lt.customer_id = 04)`: This condition excludes content items that have been listened to by a specific customer with ID 04. It ensures that the output does not include content already listened to by that particular customer.

3. Ordering and limiting:

  - `ORDER BY RAND ()`: This orders the result randomly, meaning the content items will be presented in a random order.
  - `LIMIT 5`: This limits the output to only include the first 5 rows.

The outcome will be a result set containing the content ID and title of up to 5 recently released content items that have not been listened to by the customer with ID 04. The order of the rows will be randomized.