TECHNICAL REPORT

OVERVIEW OF NORMALIZATION:

The music store database contains tables based on products(albums), clients, songs, ordered items and ratings. At first, we had a database where everything was included into 3 tables where there was many repeating rows and columns. To tackle this problem, we normalized the database to 1NF and then 2NF. We increased to number of tables in the database so that it is easier to query through it. This got rid of the repeating columns problem. Then we normalized it to 3NF with each table has single primary key where each of the other columns depend on that primary key column. This got rid of the repeating rows problem. The following are included in our database:

Tables:

- Album_ratings
- Albums
- Clients
- Genre
- Music_store
- Order_items
- Rent_items
- Song
- Song_ratings

Triggers:

- Trigger_new_client_order_items
- Trigger_new_client_rent_items
- Trigger_product_remaining
- Trigger_rent_product_remaining

Events:

- Event_restocking

Procedure:

- Check_rented_products
- Check_returned_products

View:

- Music_store_view

GENERAL SCENARIO:

Our music store covers information about the clients, the albums, when the client has ordered it, is it an ordered item or an rented item these types of basic situations. We tackled some common questions like:

QUERY 1: Display a list of clients that spent more than the average spent by client in the past month

- SELECT first_name, last_name FROM clients c JOIN order_items o ON c.client_id = o.client_id JOIN albums a ON o.album_id = a.album_id WHERE (o.quantity*a.item_price) > (SELECT AVG(o.quantity*a.item_price) FROM order_items o JOIN albums a ON o.album_id = a.album_id) ORDER BY first_name,last_name;

QUERY 2: The top sold products and least sold products over a week.

- select max(total) AS "Maximum sold products", min(total) AS "Minimum sold products" from (select album_id,SUM(quantity) as total from order_items where order_date between date_sub(now(),INTERVAL 1 WEEK) and now(group by album_id) t;

Query 3: The maximum price of products in the same category (for example, rock, pop, country, hip-hop). Use group by to list all the categories and their maximum price

- select genre.genre, max(albums.item_price) AS "Maximum price" from genre,albums where genre.genre_id=albums.genre_id
  group by albums.genre_id;

Query 4: List how many customers the system has by location (Country, Province, and City), and then sort them

- select country,province,city,count(client_id) as "Number of clients" from clients group by country,province,city order by country,city,province;

Query 5: List how many products the store has sold for a particular month

- select album_id,sum(quantity) as "Product sold" from order_items where month(order_date)=2 GROUP BY album_id;

Query 6: List how many distinct albums each singer has

- SELECT singer_name,COUNT(DISTINCT album_id) as "Distinct_albums" FROM song s GROUP BY singer_name order by Distinct_albums DESC;

Query 7: List how many copies of an album are available of a particular singer.

- SELECT DISTINCT a.album_id,album_name, singer_name, product_remaining AS "ALBUMS REMAINING"FROM albums a JOIN song s ON a.album_id = s.album_id;

Specific Scenario:

In our database we tried to fix common issues with any database system that is to prevent from manually changing each data which is connected directly or indirectly with other data.

- More Dynamic: We introduced in our database important triggers and events so that any time a change has been made to a particular table all other tables related to data is also changed.

- New Elements: We added a rating feature related to each songs and albums for the users so that before buying a product they can know about the popularity and their demand.

- Procedural response: We created procedures so that any time a client returns a product or the time has expired, the database just has to call a function rather than updating each table manually.

We created three specific scenarios with the new elements:

QUERY 1: Average rating of a singer's songs available in the store in the store

select singer_name, count(song_id) as "total songs", ROUND (avg(rating),2) as "Average rating" from music_store_view group by singer_name;

QUERY 2: All songs available for an album with ratings

select singer_name,album_name,song_name,rating from albums a JOIN song s ON a.album_id = s.album_id JOIN song_ratings sr ON s.song_ratings_id = sr.song_ratings_id order by singer_name;

QUERY 3: How many songs are there in an album with album rating

select singer_name,count(song.album_id) as "Number of songs",album_name,rating from song,albums,album_ratings where

album_ratings.album_ratings_id=albums.album_ratings_id and song.album_id=albums.album_id group by singer_name,song.album_id order by count(song.album_id);

LIST OF CONSTRANTS:

Albums-:

	Field_Name	Type (Size)	Constraints
1	Album_id	Int	Primary Key
2	Genre_id	Int	Foreign Key
3	Album_raiting_id	Int	Foreign Key
4	Store_id	Int	NOT NULL
5	Album_Name	Varchar(45)	NOT NULL
6	Label	Varchar(45)	NOT NULL
7	Product_remaining	Int	NOT NULL
8	Item_Price	Decimal	NOT NULL
9	Rent_Price	Decimal	NOT NULL

Album_Rating-:

S.No	Field_Name	Type (Size)	Constraints
1	Album_rating_id	Int	Primary Key
2	Rating	Int	

Genre-:

S.No	Field_Name	Type (Size)	Constraints
1	Genre_id	Int	Primary Key
2	Genre	Varchar(45)	NOT NULL

Clients-:

	Field_Name	Type (Size)	Constraints
1	Client_id	Int	Primary Key
2	First_name	Varchar(45)	NOT NULL
3	Last_name	Varchar(45)	NOT NULL
4	Email_address	Varchar(45)	NOT NULL, UNIQUE
5	Country	Varchar(45)	NOT NULL
6	Province	Varchar(45)	NOT NULL
7	City	Varchar(45)	NOT NULL

Music_Store-:

S.No	Field_Name	Type (Size)	Constraints
1	Store_id	Int	Primary Key
2	Store_name	Varchar(45)	NOT NULL

Order_items-:

	Field_Name	Type (Size)	Constraints
1	item_id	Int	Primary Key
2	Order_date	Datetime	NOT NULL
3	Album_id	Int	Foreign Key
4	Quantity	Int	NOT NULL
5	Client_id	Int	Foreign Key

Rent_items-:

	Field_Name	Type (Size)	Constraints

1	rent_id	Int	Primary Key
2	Rent_date	Datetime	NOT NULL
3	Album_id	Int	Foreign Key
4	Quantity	Int	NOT NULL
5	Client_id	Int	Foreign Key
6	Return_Date	Datetime	NOT NULL
7	current_status	Varchar(45)	NOT NULL

Song_Rating-:

S.No	Field_Name	Type (Size)	Constraints
1	Song_rating_id	Int	Primary Key
2	Rating	Int	

Song-:

	Field_Name	Type (Size)	Constraints
1	Song_id	Int	Primary Key
2	Album_id	Int	Foreign Key
3	Song_Rating_id	Int	Foreign Key
4	Genre_id	Int	Foreign Key
5	Song_name	Varchar(45)	NOT NULL
6	Singer_name	Varchar(45)	NOT NULL