**MUSIC STORE DATABASE**

**TECHNICAL REPORT**

**Index:-**

1. **Introduction : The concept and overview**
2. **Schema Design and Overview, Modelling, Database objects, constraints and relationships among entities**
3. **Addressing Queries and Case Scenarios: both common queries and custom queries for our group**
4. **Scope for further enhancement of the project and ‘Way Forward’**
5. **Introduction : The concept and overview**

The ‘Music Store Database’ project implemented by our group aims to model a database for a physical music store.

Breakdown of tasks and sub-tasks towards the proposed solutions :-

1. Our group identified the scope of the project, understood precise requirements, planned the overall architecture and design, and began with rough sketches, prototypes models.

1. Further, we discussed the various database elements, and how they would be organized to form tables, rows and columns; how mapping, inter-linking among tables through joins should be implemented; the need of various constraints and enforcing consistency, referential integrity of the database
2. Through continuous discussions/ group meetings and iterative refinements,

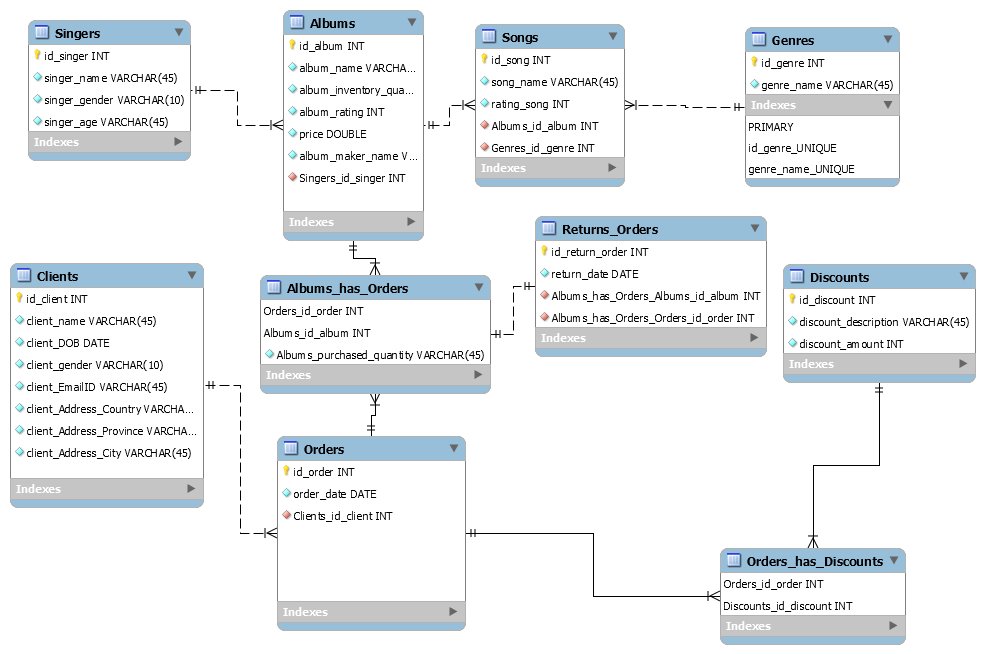
we proceeded with finalizing on an UML (Unified Model Language) diagrammatic representation, and used the ‘MySql’ workbench tool to develop the UML diagram for our project.

1. Using the UML diagram as the blue print, the group created the ‘create database’ script, using the ‘forward reengineer/ create script’ feature of MySql.
2. As the next step, the ‘create database’ script was implemented, ‘data-enteries’ into relevant database objects was done, the database was backedup, and a script was saved, that included both the database structure as well as data

(6)Finally, the required queries, and our project specific case scenarios were implemented through sql queries; and all results, outputs, screenshots were documented.

1. **Schema Design, Overview and Implementation**

The following UML diagram represents the proposed data-model.



**Key database objects used to model the database:-**

* The **‘Customers or Clients’** table contains the data about all clients who have made a transaction, and purchased any product from the ‘Music Store’. Here, data elements like client data of birth field can be used to provide special ‘birthday discount’ to the client, in case she/ he purchases any product on her/ his birthday.
* The ‘Music Store Database’ maintains, an inventory of all available albums. This inventory information is stored in the **‘Inventory or Albums’** table. Each album can have only one singer, but a singer can sing multiple albums. The singer details are stored in a separate , normalized **‘Singers’** table
* Further, each Album has can have multiple songs, but one specific song can belong to only one Album. The details of each song, are stored in a specific normalized **‘Songs’** table. Also, there are different categories or genres to which a song can belong, and these details are stored in a different ‘**Genres**’ table. Examples :- **Romance, Action, Patriotism etc.** A song can have only one specific genre but one genre can contain multiple songs.
* Our ‘Music Store Database’ implements a discount policy system. The details of various discounts are stored in a separate ‘**Discounts**’ table. A client order can have either no discount, a specific discount, or a combination of one or more discounts applied. Each discount application, reduces the client’s bill.
* The ‘**Orders/ Sales**’ table stores data of various orders made, and is the crucial table linked with discounts, albums, clients tables.

-Further, there exists a ‘**Returns\_Orders’** tables to keep track of returned products, and there are linking tables called ‘Orders\_has\_Discounts’ , ‘Albums\_has\_Orders’ to implement many-to-many relationships. Example :- The same order can have many different discounts and the same discount-type can be applied to many different orders. Similarly, an order can have many albums, and the same album can be associated with many different orders.

1. **Addressing Queries and Case Scenarios: both common queries and custom queries for our group**

* The queries formulated to address the 7 common questions are provided as an attachment to the submitted project folder

**-For our 3 specific case scenarios, the queries were as follows:-**

**(3.1) Analysis of availed discounts : which customers are availing which all documents, and the variation/ range of applied savings**

use mydb;

drop view if exists myview;

create view myview as

(select id\_client, client\_name, id\_order, discounts\_id\_discount, discount\_description,discount\_amount as 'Discount\_Amount\_in\_CAD'

from discounts join orders\_has\_discounts on orders\_has\_discounts. Orders\_id\_order = discounts.id\_discount

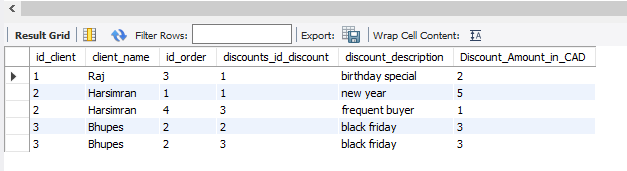
join orders on id\_order = orders\_has\_discounts.Orders\_id\_order

join clients on clients.id\_client = orders.Clients\_id\_client)

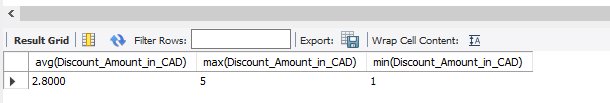
order by id\_client

;

select \* from myview;



select avg(Discount\_Amount\_in\_CAD),max(Discount\_Amount\_in\_CAD), min(Discount\_Amount\_in\_CAD) from myview;

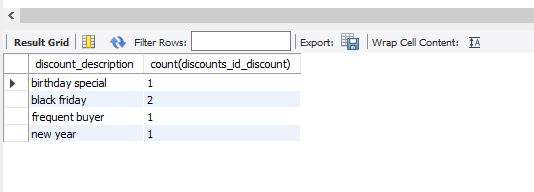


**(3.2) What factors have allowed the customers to get maximum discount, and thereby maximize his/ her savings; Due to which factors, one or more customers are losing on the available discount?**

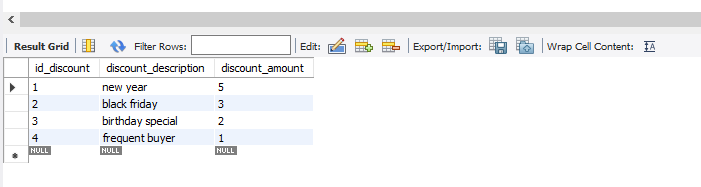
use mydb;

select discount\_description, count(discounts\_id\_discount) from orders\_has\_discounts join discounts on discounts.id\_discount = orders\_has\_discounts.Orders\_id\_order

group by Discount\_description ;



select \* from discounts;



**(3.3) What factors have allowed the customers to get maximum discount? Analysis of how some customers are availing multiple discounts and how some customers are losing on discounts and potential savings; Analysing information to send feedback to customers to maximize their savings.**

use mydb;

drop view if exists myview;

create view myview as select orders\_id\_order, Discounts\_id\_discount, discount\_description, discount\_amount as 'Discount\_Amount\_in\_CAD' from orders\_has\_discounts

join discounts on discounts.id\_discount = Discounts\_id\_discount

where Orders\_id\_order in

(

select orders\_id\_order from orders\_has\_discounts

group by Orders\_id\_order

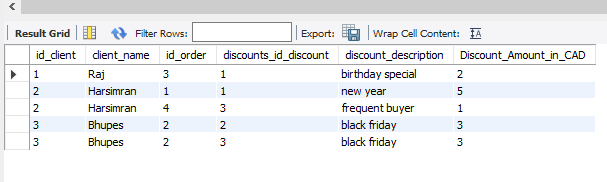
having count(Discounts\_id\_discount) >= 2

)

order by Orders\_id\_order

;

select \* from myview;



select orders\_id\_order, clients\_id\_client, client\_name, client\_EmailID, discount\_description, sum (Discount\_Amount\_in\_CAD)

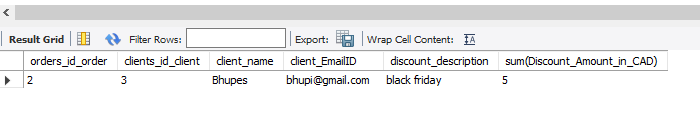
from myview join orders on orders.id\_order = myview.orders\_id\_order

join clients on clients.id\_client = orders.Clients\_id\_client

group by orders\_id\_order

order by orders\_id\_order

;



1. **Scope for further enhancement of the project and ‘Way Forward’**

* Triggers can be implemented to auto-populate the discounts table whenever a matching discount is found.

Example :- providing new year discount for all orders made on 1January of any year

* Atomic blocks of code can be implemented as a transaction, with commit and rollback features.

Example :- Whenever an order qualifies for a discount, an entry in orders\_has discount table should not be made, unless a corresponding entry in orders table is made, and vice-versa.

* Indexes can be applied to those columns, which are commonly fetched but rarely updated. This will speed up the database querying speed and efficiency.

**Conclusion :** Withiterative refinements and continuous database optimization, the database can be made more robust, scalable, efficient and responsive. Our proposed solution aims to meet the project requirements accurately, at the same time leaving scope for future adjustments.