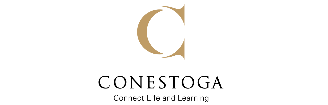
****

**CONESTOGA COLLEGE**

**DATABASE DESIGN AND MAINTENANCE**

PROG 8320

**Final Group Project: Music Store Database**

**Submitted to:**

Prof. Le Nguyen

**Group 4:**

Diego Fernando Carreno Canizares (8749994)

Sanjay Kunchappan (8749977)

Arman Singh (8750429)

**Introduction**

The goal of this project is to develop a database for a Music Store. This database must record customer information, order and payment information, and an inventory of all products sold in the Music Store. As a result, the functionality in this context is to insert, update, and remove records for the various business entities.

The database is designed for the Music Store's staff and managers.

Customers can order online through the Music Store's website, or they can order in person.

The primary requirements would be:

• Add, delete or modify Customer information

• Add, delete or modify information about the album, artists and songs.

• Add, delete or modify information about instruments.

• Insert new and modify orders and order details.

• Ability to arrange information for the store manager in terms of which customers spent more money than the average.   
This enables the manager to get a greater understanding about the customers.

• The customers are able to see the top selling items ranking and hence would be able to make appropriate purchase decisions.

**Business Scenarios**

### Problem

* + All clients can buy/sell an article (music/movie or instrument). The system should know if an article is available on the inventory and keep audits for all the purchases.
  + The employees can not be clients of the store.

### Solution

* + We will create a database system that can store the records about clients, orders with their buying items, and related information about employees.
  + We add the article entity with the client’s foreign key for a (many-to-1) relationship (1 article can be assigned to 1 client, 1 client can have many articles). We only keep the articles the store owns. We keep track of the orders and inventory separately.
  + We algo will Include combos. i.e. combine an article along with an instrument into a single combo.

**General Scenarios**

Queries for the following general scenarios were written as part of the project:

-- 1. **Display a list of clients that spent more than the average spent by clients in the past month.**

SELECT DISTINCT first\_name,last\_name,email FROM clients a, contacts co,orders c,ordersdetail d

WHERE a.contact\_id = co.contact\_id

AND a.client\_id = c.client\_id

AND c.order\_id = d.order\_id

AND price > (

SELECT AVG(price) FROM ordersdetail a, orders b

WHERE a.order\_id = b.order\_id

AND date\_format(order\_date,'%Y-%M')= '2021-November');

-- 2. **The top sold and least sold instruments over a month.**

SELECT name AS instrument\_name,quantity AS "most quantities sold" FROM orders a, ordersdetail b, instruments c

WHERE a.order\_id = b.order\_id

AND b.instrument\_id = c.instrument\_id

AND c.instrument\_id IS NOT NULL

AND date\_format(order\_date,'%Y-%M')= '2021-October'

ORDER BY quantity DESC

LIMIT 1;

SELECT name AS instrument\_name,quantity AS "least quantities sold" FROM orders a, ordersdetail b, instruments c

WHERE a.order\_id = b.order\_id

AND b.instrument\_id = c.instrument\_id

AND c.instrument\_id is not null

AND date\_format(order\_date,'%Y-%M')= '2021-October'

ORDER BY quantity ASC

LIMIT 1;

-- 3. **The maximum price of products in the same genre (for example, rock, pop, country, hip-hop). Use GROUP BY to list all the genres AND their maximum price.**

SELECT genre,max(price) FROM tracks a, inventory b

WHERE a.track\_id = b.track\_id

GROUP BY genre;

-- 4. **List how many customers the system has by location (Country, Province, AND City), AND then sort them.**

SELECT first\_name,last\_name,email,name as province\_name,city\_name FROM clients a, contacts co, address b , cities c, provinces d

WHERE a.contact\_id = co.contact\_id

AND a.address\_id = b.address\_id

AND b.city\_id = c.city\_id

AND c.province\_id = d.province\_id

ORDER BY name,city\_name;

-- 5. **List how many quantities the store has sold for a particular month.**

SELECT sum(quantity) FROM orders a, ordersdetail b

WHERE a.order\_id = b.order\_id

AND date\_format(order\_date,'%Y-%M')= '2021-October';

-- 6. **List how many DISTINCT albums each singer has.**

SELECT artist,count(DISTINCT title) FROM albums a, tracks b

WHERE a.album\_id = b.album\_id

GROUP BY artist;

-- 7. **List how many copies of an album are available of a particular singer.**

SELECT artist,name as album\_name,title,quantity FROM albums a

INNER JOIN tracks b

ON a.album\_id = b.album\_id

INNER JOIN inventory c

ON b.track\_id = c.track\_id;

**Specific Scenarios**

-- **QUERY 1 : User wishes to view ranking of items in terms of most items sold after a specific date.**

-- Instruments sold the most after October 27, 2021

-- Tracks sold the most after October 27, 2021

-- Tracks and Instruments sold the most after October 27, 2021 within Combos

USE music\_store\_db;

-- PARTITION BY B.Type, B.overall\_quantity

SELECT \*,DENSE\_RANK() OVER(ORDER BY B.overall\_quantity DESC) as TheRank

FROM (

SELECT A.Type,A.Item\_Name,SUM(quantity) overall\_quantity

FROM

(

SELECT o.order\_id, o.order\_date, o.store\_id, od.quantity,

IF (od.track\_id IS NOT NULL, t.title, i.name) as 'Item\_Name',

IF (od.track\_id IS NOT NULL, 'TRACK', 'INSTRUMENT') as 'Type'

FROM Orders o

INNER JOIN OrdersDetail od ON o.order\_id = od.order\_id

LEFT JOIN Tracks t ON od.track\_id = t.track\_id

LEFT JOIN Instruments i ON od.instrument\_id = i.instrument\_id

WHERE od.combo\_id IS NULL

UNION

SELECT o.order\_id, o.order\_date, o.store\_id,od.quantity,

i.name as 'Item\_Name',

'INSTRUMENT' as 'Type'

FROM Orders o

INNER JOIN OrdersDetail od ON o.order\_id = od.order\_id

LEFT JOIN CombosHasInstruments chi ON od.combo\_id = chi.combo\_id

LEFT JOIN Instruments i ON chi.instrument\_id = i.instrument\_id

WHERE od.combo\_id IS NOT NULL

UNION

SELECT o.order\_id, o.order\_date, o.store\_id,od.quantity,

t.title as 'Item\_Name',

'TRACK' as 'Type'

FROM Orders o

INNER JOIN OrdersDetail od ON o.order\_id = od.order\_id

LEFT JOIN CombosHasTracks cht ON od.combo\_id = cht.combo\_id

LEFT JOIN Tracks t ON cht.track\_id = t.track\_id

WHERE od.combo\_id IS NOT NULL

) A

WHERE order\_date >= '2021-10-27'

GROUP BY A.Type,A.Item\_Name

ORDER BY A.Type,A.Item\_Name

) B

ORDER BY B.Type,TheRank

;

-- **Query 2 : Customer details along with last purchase date for specific email.**

SELECT first\_name,last\_name,email,phone\_no,address\_1,address\_2,zip\_code,city\_name,name as province\_name,city\_name,MAX(order\_date) as last\_purchase\_on

FROM clients a, contacts co, address b , cities c, provinces d, orders e

WHERE a.contact\_id = co.contact\_id

AND a.address\_id = b.address\_id

AND b.city\_id = c.city\_id

AND c.province\_id = d.province\_id

AND a.client\_id=e.client\_id

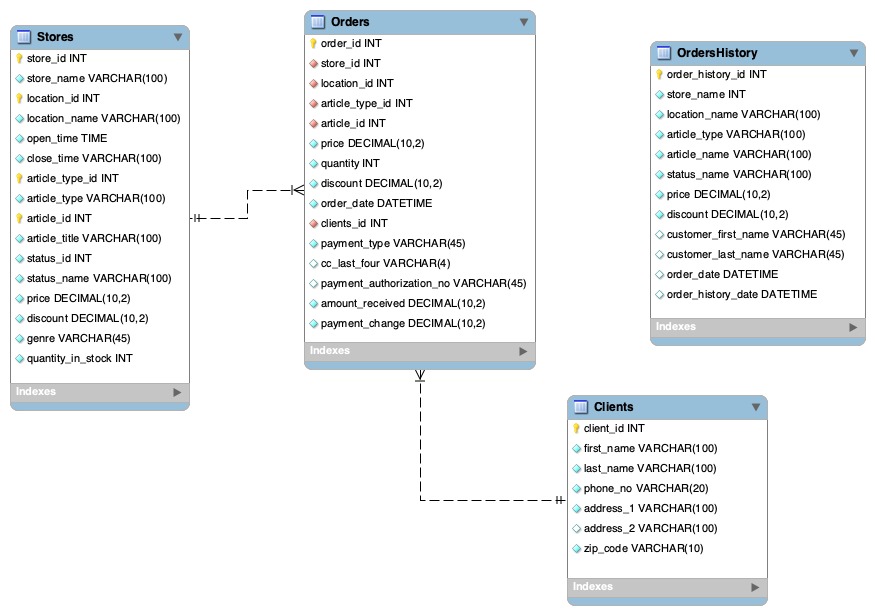
AND email='PSTARK@gmail.com';

**Documentation**

### **ER MODELS:** 1.0

* Created a first draft, only with the stores, orders and audit data.

Before Normalisation:



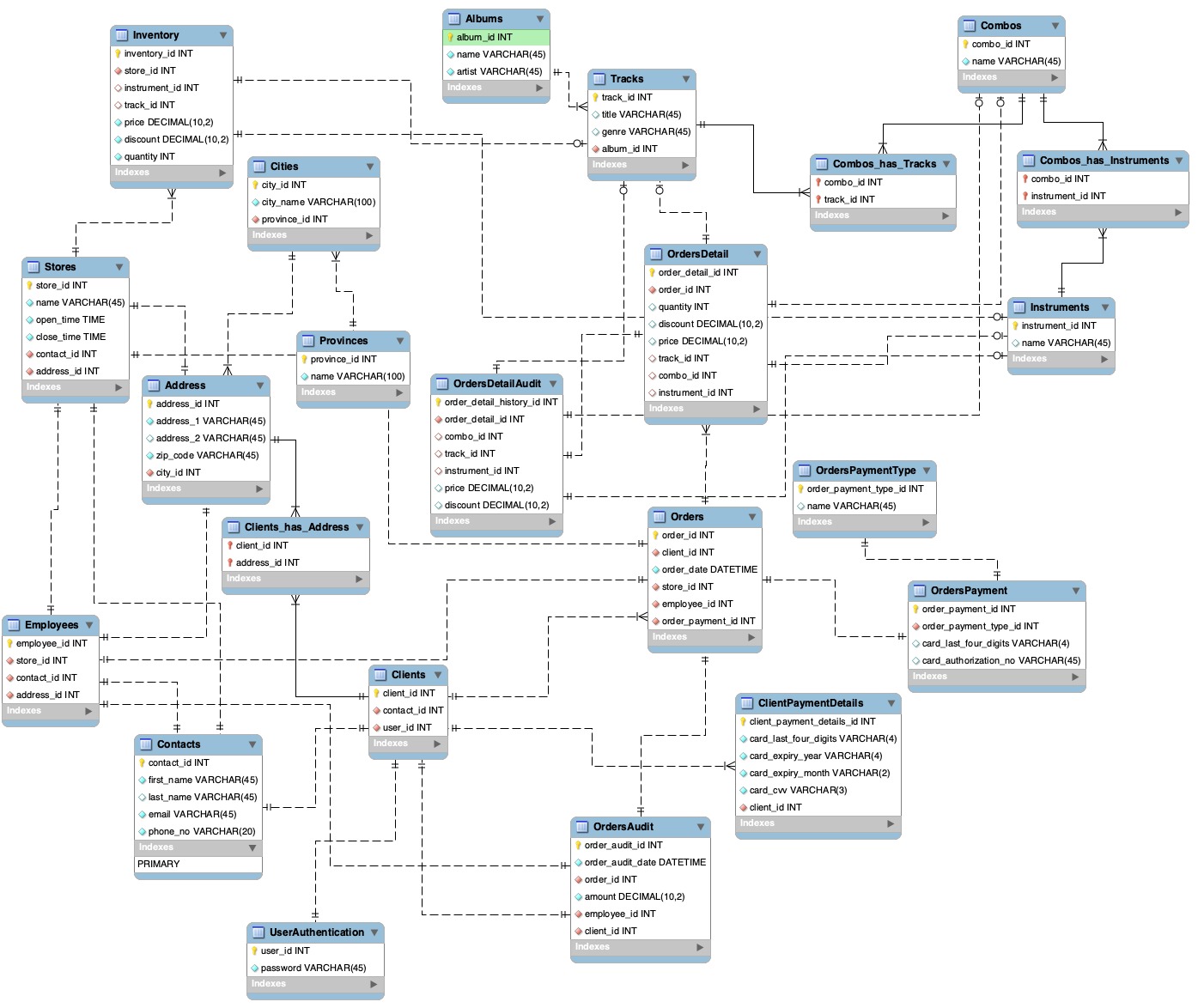
### 2.0

* + Normalizing the diagram in version 1.0
  + Splitting the stores into locations, store, contacts, clients, Inventory, articles, combos, instruments, orders with their items separately and audit table for the orders and ordersdetail.

### 3.0

* + Adding the employee first name and last name on the employee’s table.
  + Create the addresses table and link it to the employee, client, and stores.
  + Remove the address columns from the contact’s table.
  + Change the articles-Inventory from one-to-many to one-to-one.
  + Change the instruments-Inventory from one-to-many to one-to-one.
  + Rename the OrdersHistory to OrdersAudit and OrdersDetailHistory to OrdersDetailAudit.
  + Adding client\_id into OrdersAudit
  + Removing all the value columns on the OrdersAudit and OrdersDetailsAudit so they reference the primary keys instead.
  + Link the payment method to the client with a one-to-one relationship.

After Normalisation:



Normalisation is a technique of organising data in a database.   
It helps in reducing redundancy and improves the data integrity of a database.  
Normalisation usually involves breaking down of large tables into smaller ones in order to make them more efficient.

**Constraints for table**

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Primary Key | Foreign Key | Other Constraints |
| address | address\_id |  | address\_id: AI |
| albums | album\_id |  | album\_id: AI |
| cities | city\_id | province\_id | city\_id: AI |
| clientpaymentdetails | client\_payment\_details\_id | client\_id | client\_payment\_details\_id: AI |
| clients | client\_id | contact\_id, user\_id, address\_id | client\_id: AI |
| clienthasaddress | client\_id, address\_id |  |  |
| combos | combo\_id |  |  |
| comboshasinstruments | combo\_id, instrument\_id |  |  |
| combohastracks | combo\_id, track\_id |  |  |
| contacts | contact\_id |  | contact\_id:AI |
| employees | employee\_id | store\_id, contact\_id, address\_id | employee\_id: AI |
| instruments | instrument\_id |  |  |
| inventory | inventory\_id | store\_id, instrument\_id, track\_id | inventory\_id: AI |
| orders | order\_id | client\_id, store\_id, employee\_id, order\_payment\_id | order\_id:AI |
| ordersaudit | order\_audit\_id | order\_id, employee\_id, client\_id | order\_audit\_id: AI |
| ordersdetail | order\_detail\_id | order\_id, track\_id, combo\_id, instrument\_id | order\_detail\_id: AI |
| ordersdetailaudit | order\_detail\_history\_id | order\_detail\_id, combo\_id, track\_id, instrument\_id | order\_detail\_history\_id |
| orderspayment | order\_payment\_id | order\_payment\_type\_id | order\_payment: AI |
| orderpaymenttype | order\_payment\_type\_id |  | order\_payment\_type\_id: AI |
| provinces | province\_id |  | province\_id: AI |
| stores | store\_id | contact\_id, address\_id | store\_id: AI |
| tracks | track\_id | album\_id | track\_id: AI |
| userauthentication | user\_id |  | user\_id: AI |

**Conclusion**

As part of this project, a database for a Music Store was developed keeping all the business scenarios in mind. This database records vital data such as customer information, order and payment information, and an inventory of all products sold in the Music Store.

A robust database was created for the client along with queries for the management to gain an understanding of the sales and customer behaviour.  
Queries were also created keeping the customers in mind to provide insights to the customer regarding popular products as well as retrieving previous orders.

Triggers were added in order to ensure data integrity.

The database was hence created in accordance with all the client requirements and delivered in a timely manner.

**Appendix**

**Table Breakdowns:**  
  
**Arman Singh:**

Inventory: In this table we have inventory\_id as a primary key and store\_id as foreign key. This table also includes the id of instruments and tracks as well as the price, discount and quantity column. This table is also linked to the stores, tracks and instrument tables.

Stores: This table has a primary key as store\_id and two foreign keys as contact\_id, addresss\_id. It also has three columns which are name, open and close. It is linked to the inventory, employees, contacts, addresses and orders table.

**Sanjay Kunchappan:**

Clients Table:

Normalisation was performed on the Clients table and it was broken down into several different tables in order to make the database more efficient and enhance data integrity.

After Normalisation, the client table was broken down into clients table, which contains client\_id as its primary key as well as contact\_id and user\_id as foreign keys.

This table contains the key client\_id which can be used to identify each individual client.

Clients table is linked to multiple tables such as orders, contacts, address, userauthentication and clientpaymentdetails.

Contacts Table:

Contacts Table contains the contact details of clients, employees and the store. It has a one-to-one relationship with the clients, employees and the store respectively. It contains the fields of contact\_id as primary key, first\_name, email and phone\_number as not nulls and last name as a non-mandatory field.

**Diego Carreno:**

Address Table:  
Address table has been broken down into address, cities and provinces table after normalisation in order to improve data integrity and enhance performance of the database.

The primary key of the address table is address\_id and each individual address can be identified by this key.

The other fields include address\_1 and address\_2 for address lines 1 and 2 respectively.

And zip code as well as a mandatory field NOT NULL field.

This table also contains the foreign key city\_id linking it to the cities table.

OrdersPayment Table:

This table contains Order Payment details pertaining to each order.

order\_payment\_id is the primary key for this table.

This table also contains a foreign key i.e. order\_payment\_type\_id which links this table to the OrderPaymentsTypeTable.

This table contains two more cells named as card\_last\_four\_digits and card\_authorisation\_no.