Andre Hei Wang Law

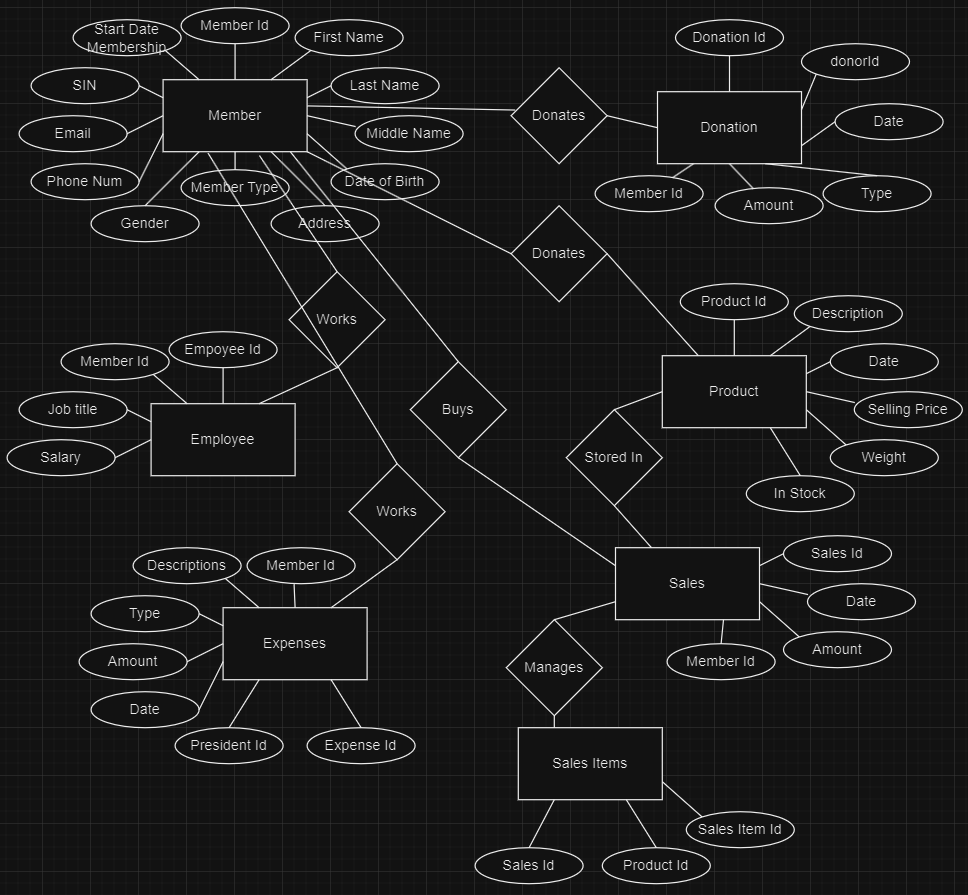
4017 5600

L\_HEIWAN

Comp 353 – Assignment 2

**PART I:**

**a) ER Diagram**



**b) Constraints Not Captured by ER Diagram**

Primary Keys:

-Member Id

-Donation Id

-Product Id

-Sales Id

-Sales Items Id

-Employee Id

-Expenses Id

Foreign Keys:

-In Donation, Member Id references to Member Id from Member

(Member Id here is the same as Donor Id)

-In Sales, Member Id references to Member Id from Member

-In Sales Items, Product Id references to Product Id from Product

-In Sales Items, Sales Id references to Sales Id from Sales

-In Expenses, President Id references to Member Id from Member

-In Expenses, Member Id references to Member Id from Member

-In Employee, Member Id references to Member Id from Member

Cardinalities:

One Member(donor) can make multiple Donations (1-to-many)

One Member(donor) can donate multiple Products (1-to-many)

One Member(client) can buy multiple Sales (1-to-many)

One Sale can have multiple Sales Items (1-to-many)

One Employee(employee) works as one Member (1-to-1)

One Member(president) can have many Expenses (1-to-many)

Many Products can be stored into one Sales (many-to-1)

**c) Relational Database Scheme**

|  |  |
| --- | --- |
| Member | |
| memberId (PK) | INT |
| firstName | VARCHAR(255) |
| lastName | VARCHAR(255) |
| middleName | VARCHAR(255) |
| dateOfBirth | DATE |
| address | VARCHAR(255) |
| memberType | ENUM(‘client’, ‘employee’, ’donor’) |
| gender | VARCHAR(255) |
| phoneNumber | VARCHAR(255) |
| email | VARCHAR(255) |
| socialInsuranceNumber | VARCHAR(255) |
| startDateMembership | DATE |

|  |  |
| --- | --- |
| Donation | |
| donationId (PK) | INT |
| donarId | INT |
| date | DATE |
| type | ENUM(‘product’, ‘money’) |
| amount | INT |
| memberId (FK) | INT |

|  |  |
| --- | --- |
| Product | |
| productId (PK) | INT |
| description | VARCHAR(255) |
| date | DATE |
| sellingPrice | INT |
| weight | FLOAT |
| inStock | BOOLEAN |

|  |  |
| --- | --- |
| Sales | |
| salesId (PK) | INT |
| date | DATE |
| amount | INT |
| memberId (FK) | INT |

|  |  |
| --- | --- |
| Sales Items | |
| salesItemId (PK) | INT |
| productId (FK) | INT |
| salesId (FK) | INT |

|  |  |
| --- | --- |
| Expenses | |
| expenseId (PK) | INT |
| presidentId | INT |
| date | DATE |
| amount | INT |
| type | ENUM(‘rent’, ‘bill’, ‘charity’) |
| description | VARCHAR(255) |
| memberId (FK) | INT |

|  |  |
| --- | --- |
| Employee | |
| employeeId (PK) | INT |
| jobTitle | ENUM(‘president’, ‘vice-president’, ‘cashier’, ‘other’) |
| salary | INT |
| memberId (FK) | INT |

**d) Changed Design – President and Vice-President Approval**

If we need the approval of both the president and the vice president of the organization, we would need to change the Expenses table. Rather than President Id, it would be Approver Id. This means, this stores the Id of the person who approved of the expense. In this case, we can have it referenced as a foreign key to Member table which needs access to Employee. This is due to the fact that Employee tables stores the Job Title of the member which includes President and Vice President.

**PART II:**

a)

**SELECT**   
 m.memberId,  
 m.firstName,  
 m.lastName,  
 m.middleName,  
 m.dateOfBirth,  
 m.address,  
 m.gender,  
 m.phoneNumber,  
 m.email,  
 m.socialInsuranceNumber,  
 m.startDateMembership,  
 e.jobTitle  
**FROM**   
 Member m -- 'Member' **as** 'm'  
**JOIN**  
 Employee e **ON** m.memberId = e.memberId -- 'Employee' **as** 'e' **on** **matching** member ID  
**WHERE**   
 m.memberType = 'client' **AND**-- **select** members **with** a memberType **of** 'client'   
 m.memberId **IN** ( -- filter members   
 **SELECT** memberId   
 **FROM** Donation -- retrieve member IDs **from** the table 'Donation'  
 INTERSECT -- INTERSECT **operator**, combine the results   
 **SELECT** memberId   
 **FROM** Employee -- retrieve member IDs **from** the table 'Employee'  
 **WHERE** salary **IS** NULL **OR** salary = 0 -- salary **is** NULL **or** 0  
 );

b)

**SELECT**  
 e.expenseId,  
 m.firstName,  
 m.lastName,  
 e.date,  
 e.amount,  
 e.type,  
 e.description  
**FROM**  
 Expenses e -- 'Expenses' **as** 'e'  
**JOIN**  
 Member m **ON** e.presidentId = m.memberId -- 'Member' **as** 'm' **on** matching president IDs  
**WHERE**  
 EXTRACT(MONTH **FROM** e.date) = 6 **AND**-- filter, result **in** the month of June = 6  
 EXTRACT(YEAR **FROM** e.date) = 2023; -- filter, result from the year 2023

c)

**SELECT**  
 s.salesId,  
 s.date,  
 m.firstName,  
 m.lastName,  
 p.description,  
 p.sellingPrice,  
 p.weight  
**FROM**  
 Sales s -- 'Sales' **as** 's'  
**JOIN**  
 Member m **ON** s.memberId = m.memberId -- 'Member' **as** 'm' **on** **matching** member IDs  
**JOIN**  
 SalesItems si **ON** s.salesId = si.salesId -- 'SalesItems' **as** 'si' **on** **matching** Sales IDs  
**JOIN**  
 Product p **ON** si.productId = p.productId -- 'Product' **as** 'p' **on** **matching** Product IDs  
**WHERE**  
 EXTRACT(MONTH **FROM** s.date) = 6 **AND** -- filter, **result** **in** the month **of** June = 6  
 EXTRACT(YEAR **FROM** s.date) = 2023 **AND** -- filter, **result** **from** the year 2023  
 m.address LIKE '%Brossard%' -- filter, **result** containing 'Brossard'  
**ORDER** **BY**  
 s.salesId **ASC**, m.firstName **ASC**, m.lastName **ASC**, p.weight **ASC**; -- **order**

d)

**SELECT**  
 s.salesId,  
 s.date,  
 p.description,  
 p.sellingPrice,  
 p.weight  
**FROM**  
 Sales s -- 'Sales' **as** 's'  
**JOIN**  
 SalesItems si **ON** s.salesId = si.salesId -- 'SalesItems' **as** 'si' **on** **matching** Sales IDs  
**JOIN**  
 Product p **ON** si.productId = p.productId -- 'Product' **as** 'p' **on** **matching** Product IDs  
**WHERE**  
 EXTRACT(MONTH **FROM** s.date) = 6 **AND** -- filter, **result** **in** the month **of** June = 6  
 EXTRACT(YEAR **FROM** s.date) = 2023 **AND**-- filter, **result** **from** the year 2023  
 **NOT** EXISTS ( -- filter based **on** abscence  
 **SELECT** 1 -- placeholder, return value 1  
 **FROM** Member m  
 **WHERE** s.memberId = m.memberId **AND**   
 m.memberType = 'client' -- member **is** a client that purchased the product  
 )  
**ORDER** **BY**  
 s.salesId **ASC**, p.sellingPrice **ASC**;

e)

**SELECT**  
 m.firstName,  
 m.lastName,  
 MIN(s.date) **AS** firstSaleDate,  
 MAX(s.date) **AS** lastSaleDate,  
 SUM(s.amount) **AS** totalAmountOfSales -- total   
**FROM**  
 Member m  
**JOIN**  
 Sales s **ON** m.memberId = s.memberId -- 'Sales' as 's' on matching Mamber IDs  
**WHERE**  
 m.memberType = 'client' -- member is a client that purchased the product  
**GROUP** **BY**  
 m.memberId, m.firstName, m.lastName  
**HAVING**  
 SUM(s.amount) >= 1000 -- at least 1000$  
**ORDER** **BY**   
 totalAmountOfSales; -- sort based on total