Gu, Shanqing

MSDS 7330 MidTerm Due Date: 07/01/2018

File Organization and Database Management Midterm Exam

Q1: The picture above is a concise representation of the various teams, format and fixtures of the 2018 soccer world cup. How would you design a database schema to hold all the relevant info pertaining to the world cup as depicted in the figure above? **(20 points)**

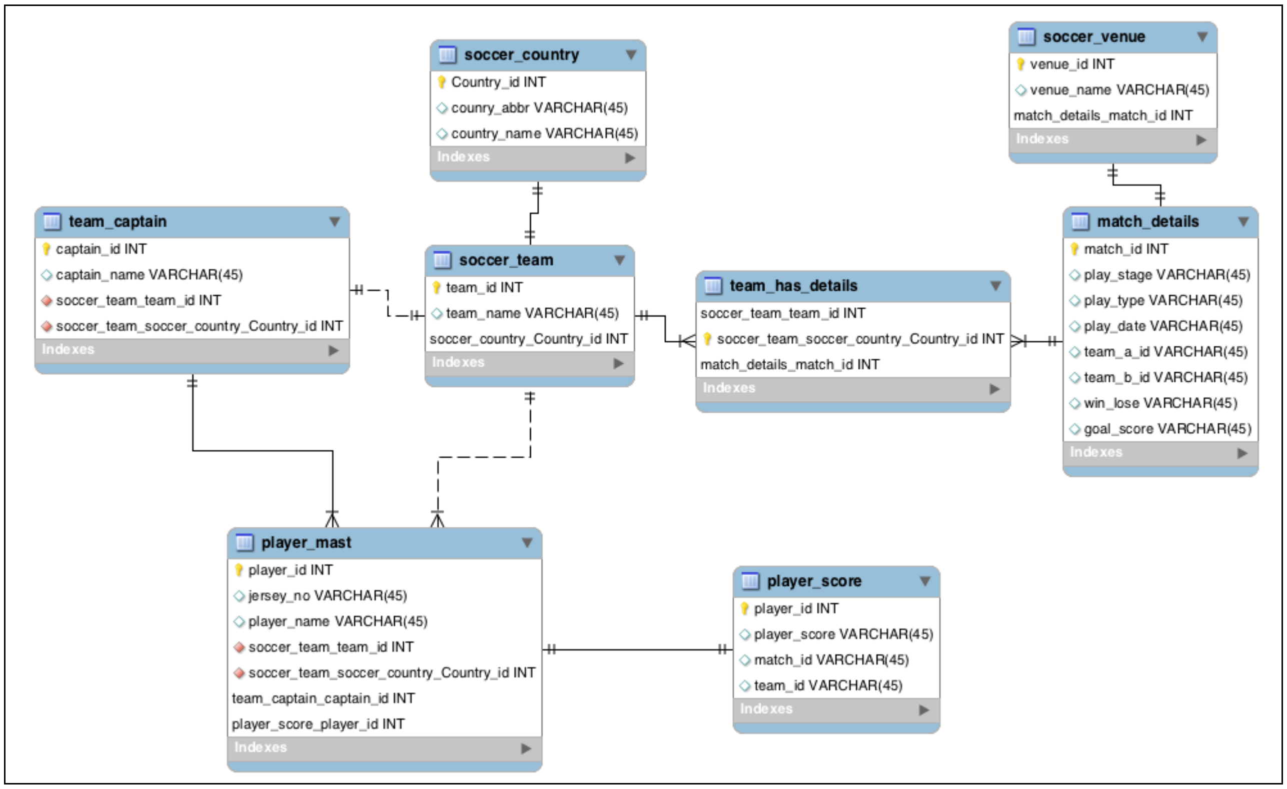
- In addition to capturing all the info in the picture into the ER model, explain how your design can address the following requirements (you can provide sample SQLs or explain query logic in plain English)

There should be a mechanism to capture all the games played by a given team
capture all the games at a given venue
captures the players for each team with their captains
capture the scores for each game player

Note:

Your answer should identify the key entities and relations between them.
Ensure you state all your assumptions while building the db model.
Summarize your design in plain English sentences along with the ER model.
Insert the picture of the ER model inline with your answer (you are free to use MySQL Workbench , visio , ppt , word or any other tool create to a readable /legible ER model)

Q1.1: Database schema for 2018 soccer world cup



Q1.2: Relationship between key entities

The model is divided into three main entities: soccer_team, match_details, player_mast
The tables outside these entities are dictionaries: soccer_country, team_captain, player_score and one many-to-many relation through mideel entity team_has_details between soccer_team and matach_details.

From all the information captured from the wall chart, we can:

(1) Ho ld all the relevant information pertaining to the world cup as depicted in the figure (see Q1.3: Description of tables);
(2) capture all games played by a given team through two tables soccer_team and match_details together with middle entity table team_has_details (n:n relationship);
(3) capture all the games at a given venue through tables soccer_venue and match_details (1:1 relationship);
(4) capture the players for each team with their captains through tables soccer_team to team_captain (1:1) and to player_mast (1:n relationship);
(5) capture the scores for each game player through tables player_mast and player_score (1:1 relationship).

Q1.3: Description of tables:

soccer_country:

- country_id: a unique ID for each country
- country_abbr: the sort name of each country
- country_name: the name of each country

soccer_venue:

- venue_id: a unique ID for each venue
- venue_name: the name of the venue
- match_details_match_id: refer to the match_id in the match_details table

team_captain:

- captain_id
- captain_name
- soccer_team_team_id: refer to the team_id in the soccer_team table
- soccer_team_soccer_county_Country_id

soccer_team:

- team_id: the ID for each team.
- Team_name
- Soccer_county_County_id: Each team represents a country referencing the country_id column of soccer_country table

match_details:

- match_id – this is the unique ID for a match
- play_stage – this indicates that in which stage a match is going on, i.e. G_A to G_H for Group stage, R for Round of 16 stage, Q for Quarter final stage, S for Semi Final stage, 3P for 3rd place play off and F for Final
- play_type
- play_date – date of the match played
- team_a_id
- team_b_id
- win_lose: team either win or lose or drawn indicated by the character W, L, or D
- goal score: how many goals scored by the team

middle entity (team_has_details):

- soccer_team_team_id
- soccer_team_soccer_country_Country_id
- match_details_match_id

player_mast:

- player_id – this is a unique ID for each player
- jersey_no – the number which labeled on the jersey for each player
- player_name – name of the player
- soccer_team_team_id
- soccer_team_soccer_country_Country_id
- team_captain_captain_id
- player_score_payer_id

player_score:

- player_id – this is a unique ID for each player
- player_score
- match_id
- team_id

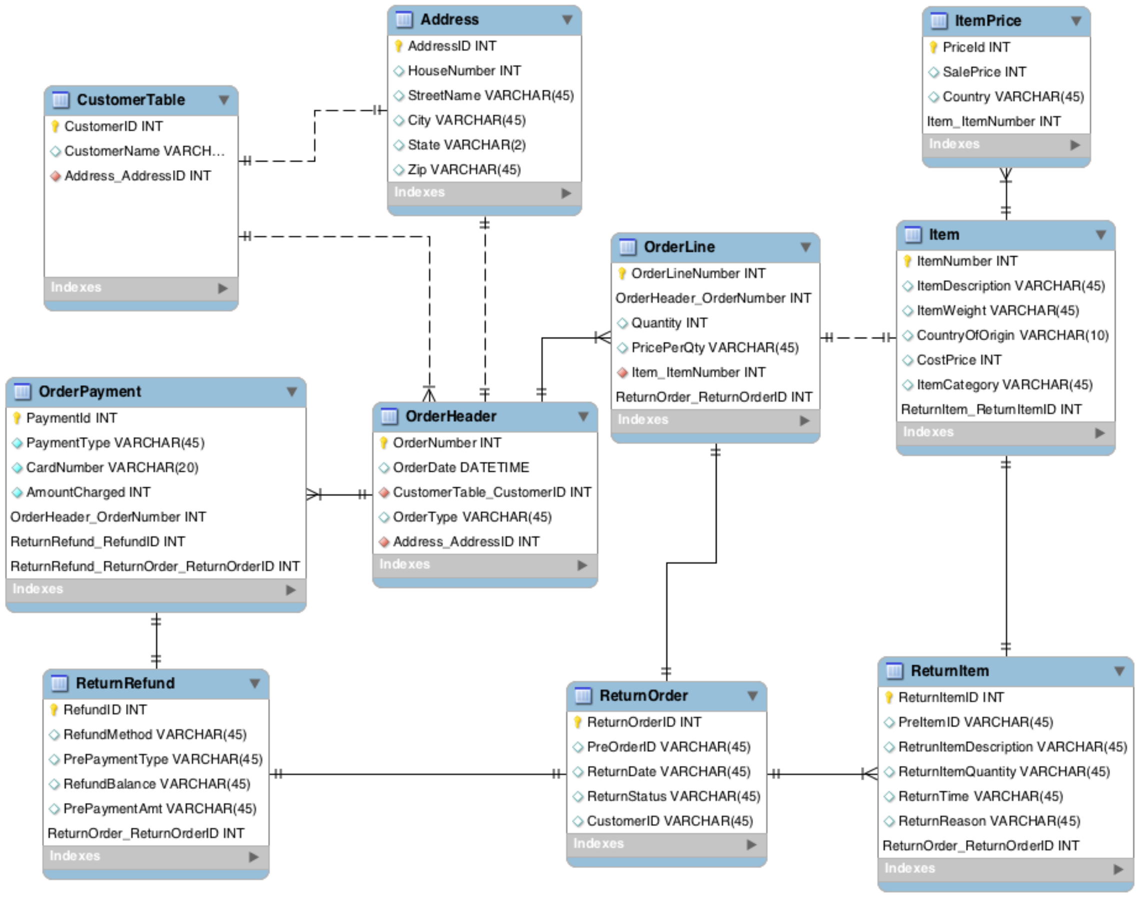
Q2: In the context of the EcommerceOrderDB we built, some customers have been complaining about the quality of the items they bought. They want to be able to return orders and get a full refund. The business has decided to support returns with the following requirements. **(20 pts)**

- Customers can return the entire order or just a part of it. Business must be able to track the items being returned with their quantities
- the system must be able to capture the refunds issued on the returns with the payment methods on which they are issued.
- To prevent fraud the business team
  - o wants to be able to track returns against the original order the customers bought
    - o ensure only the items that were originally bought
    - o ensure the returned quantity does not exceed the original quantity bought .

  what features/capabilities of databases would you use to handle such quantity validations.
- Your task is to update the design of the ecommercedb schema to handle these new reqts
- Describe how your changes impact the existing orders already in the systems.
- Change is the only constant. What are some of the strategies you recommend to future proof your design

  Note:
  - Identify the relevant entities and relations that are to be designed/ enhanced from existing structure for supporting the new requirements
  - State your assumptions clearly
  - Summarize your design approach in bullet points in addition to the ER diagram representing any new entities .
  - Insert the picture of the ER model inline with your answer (you are free to use MySQL
    Workbench , visio , ppt , word or any other tool create to a readable /legible ER model
  - You can choose to hand create sample return order record(s) to aid in better explanation of your model. The data can be mocked up in excel / notepad to better explain your design.

Q2.1: Update the design of the ecommercedb schema to handle new requirements related to return orders



Q2.2: Relationship between key entities

In order to meet the new requirements for returning orders and getting a full refund, add three new entities (ReturnRefund, ReturnOrder, ReturnItem), and set the entity relationships as shown in Q2.1.

In details:

(1) In order to know if customers return the entire order or just a part of it, the business team have to track the items being returned with their quantities. New added ReturnItem entity includes 7 attributes (ReturnItemID, PreItemID, ReturnItemDescription, ReturnItemQuantity, ReturnTime, ReturnReason, ReturnOrder_ReturnOrderID). Its one-to-one relationship with previous Item entity is set up through new added attribute ReturnItem_ReturnItemID in Item table.

(2) In order to capture the return refunds, the business team uses the new entity ReturnRefund including 6 attributes (RefundID, RefundMethod, PrePaymentType, RefundBalance, PrePaymentAmt, ReturnOrder_ReturnOrderID). The ReturnOrder_ReturnOrderID connects entities ReturnRefund and ReturnOrder. The OrderPayment table connects ReturnRefund table through ReturnRefund_RefundID and ReturnRefund_ReturnOrder_ReturnOrderID.

(3) For fraud prevention (returns against the original order, only the items that were originally bought, the returned quantity does not exceed the original quantity), the business team has to create multiple column constraints for data field validation (order, item, and quantity), such as the same CustomerID, the same itemID and the same (or less) quantity for original order and item returns/refund.

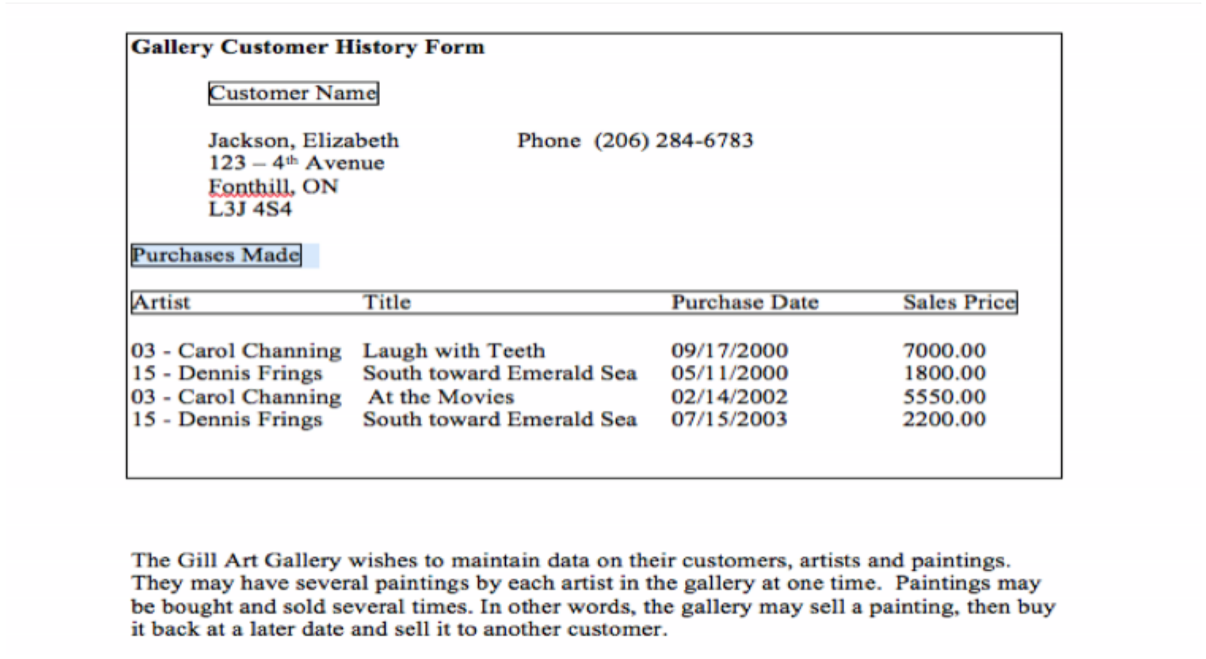
(4) The new return and refund updates have impacts on existing tables when adding new attributes to set up relationships with the corresponding tables.

(5) The strategies to adding changes for future-proof design.

There are several different kinds of changes: no impact (adding new tables), minimal impact (adding columns to existing tables with no data size change), other changes will wreak havoc on down time (any operations like index rebuild, data size changes).

Therefore, the database should be 3NF from to minimize impact from database modification, such as: no duplicate data in a single row, eliminate data subsets that are recurring multiple times, remove any data that is not directly related to the table.

Q3: **(20 pts)**



Gallery Customer History Form

Customer Name

Jackson, Elizabeth Phone (206) 284-6783
123 – 4th Avenue
Fonthill, ON
L3J 4S4

Purchases Made

Artist	Title	Purchase Date	Sales Price
03 - Carol Channing	Laugh with Teeth	09/17/2000	7000.00
15 - Dennis Frings	South toward Emerald Sea	05/11/2000	1800.00
03 - Carol Channing	At the Movies	02/14/2002	5550.00
15 - Dennis Frings	South toward Emerald Sea	07/15/2003	2200.00

The Gill Art Gallery wishes to maintain data on their customers, artists and paintings. They may have several paintings by each artist in the gallery at one time. Paintings may be bought and sold several times. In other words, the gallery may sell a painting, then buy it back at a later date and sell it to another customer.

Design the constituent entities and relations ensuring that they satisfy 1NF, 2NF and 3NF forms. You are **required** to apply the normalization techniques to arrive at the final design. The following is the UnNormalized Representation of the above data:

customer [custno, cust_name, cust_addr, cust_phone, (artist_id, artist_name, art_title, pur_date, price)]

Un-Normalized Form (UNF):

Get all the attributes, put them into a list, identify the primary key with underline, and indent the repeating groups.

customer [custno, cust_name, cust_addr, cust_phone,
                    (artist_id, artist_name, art_title, pur_date, price)]

First Normal Form (1NF):

Remove the repeating groups to a new entity cust_art, assign a code to each piece of art (art_code) as the new primary key, and keep custno and pur_date since one piece of art may be bought more than one time.

customer [custno, cust_name, cust_addr, cust_phone]
cust_art [art_code, artist_id, artist_name, art_title, price, custno*, pur_date*]

Second Normal Form (2NF):

Look at tables with a composite key, review each non-key attribute to identify if the attribute is dependent on part of the key or all the key, and remove any partial key and dependents to a new table. Here, the art_title is dependent only on artist_name. There exists partial dependency. Remove that to get the following 2NF.

customer [custno, cust_name, cust_addr, cust_phone]
cust_art [custno, art_code, pur_date, price]
art [art_code, art_title, artist_id, artist_name]

Third Normal Form (3NF):

The transitive dependency of artist_name on art_title from art relation is removed.

customer [custno, cust_name, cust_street, cust_city, cust_state, cust_postcode, cust_phone]
cust_art [custno, art_code, pur_date, price]
art [art_code, art_title, artist_id (FK)]
artist [artist_id, artist_fname, artist_lname]

Q4:

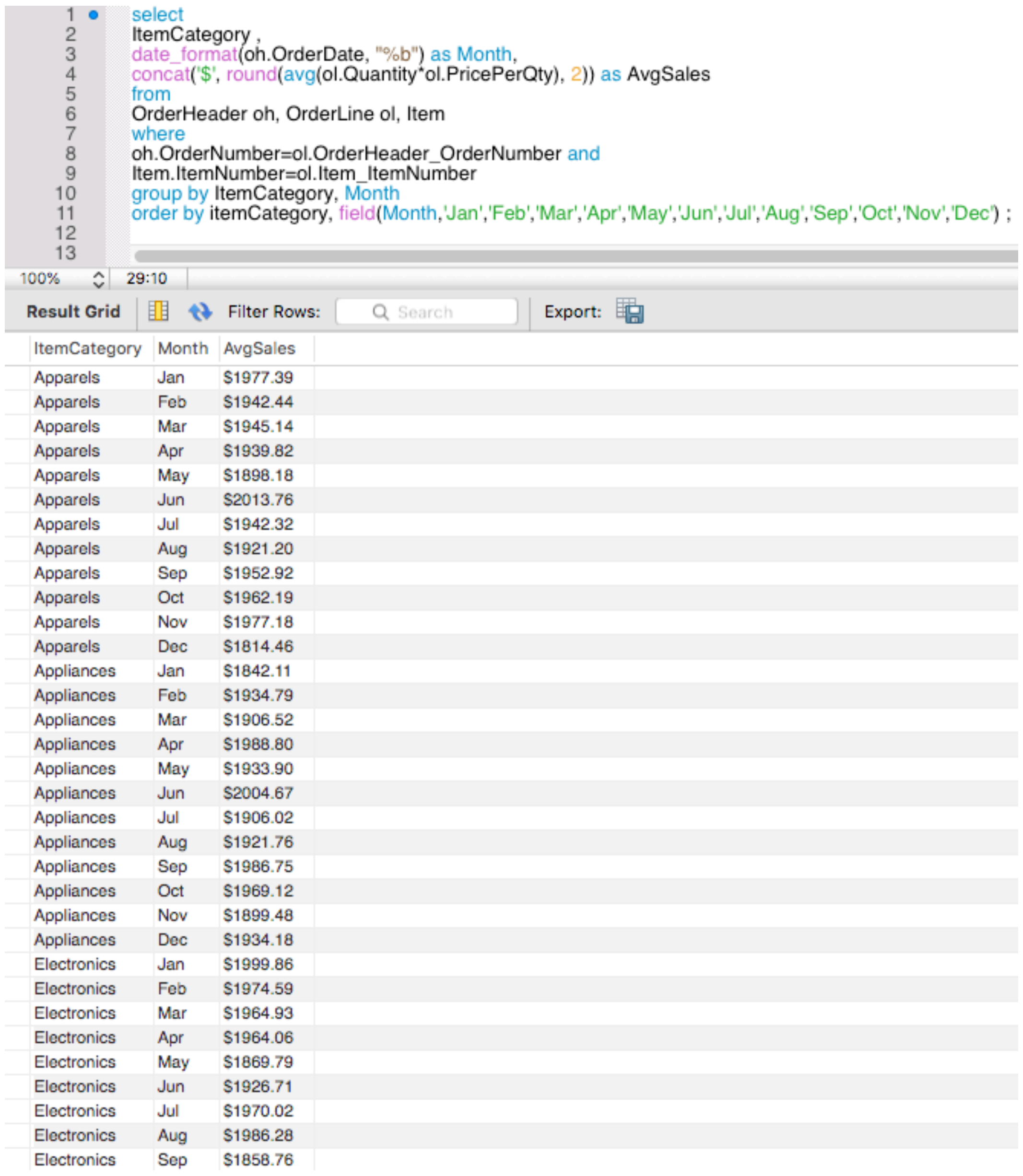
In the context of the Ecommerce Order Database we designed in the class, create SQL to address the following questions

• Calculate the average revenue in sales for each item category for each month **(10 pts)**

Q4.1.1 SQL scripts:

```
select
ItemCategory,
date_format(oh.OrderDate, "%b") as Month,
concat('$', round(avg(ol.Quantity*ol.PricePerQty), 2)) as AvgSales
from
OrderHeader oh, OrderLine ol, Item
where
oh.OrderNumber=ol.OrderHeader_OrderNumber and
Item.ItemNumber=ol.Item_ItemNumber
group by ItemCategory, Month
order by itemCategory, field(Month,'Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec') ;
```

Q4.1.2 SQL results:



```
select
ItemCategory ,
date_format(oh.OrderDate, "%b") as Month,
concat('$', round(avg(ol.Quantity*ol.PricePerQty), 2)) as AvgSales
from
OrderHeader oh, OrderLine ol, Item
where
oh.OrderNumber=ol.OrderHeader_OrderNumber and
Item.ItemNumber=ol.Item_ItemNumber
group by ItemCategory, Month
order by itemCategory, field(Month,'Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec') ;
```

ItemCategory	Month	AvgSales
Apparels	Jan	$1977.39
Apparels	Feb	$1942.44
Apparels	Mar	$1945.14
Apparels	Apr	$1939.82
Apparels	May	$1898.18
Apparels	Jun	$2013.76
Apparels	Jul	$1942.32
Apparels	Aug	$1921.20
Apparels	Sep	$1952.92
Apparels	Oct	$1962.19
Apparels	Nov	$1977.18
Apparels	Dec	$1814.46
Appliances	Jan	$1842.11
Appliances	Feb	$1934.79
Appliances	Mar	$1906.52
Appliances	Apr	$1988.80
Appliances	May	$1933.90
Appliances	Jun	$2004.67
Appliances	Jul	$1906.02
Appliances	Aug	$1921.76
Appliances	Sep	$1986.75
Appliances	Oct	$1969.12
Appliances	Nov	$1899.48
Appliances	Dec	$1934.18
Electronics	Jan	$1999.86
Electronics	Feb	$1974.59
Electronics	Mar	$1964.93
Electronics	Apr	$1964.06
Electronics	May	$1869.79
Electronics	Jun	$1926.71
Electronics	Jul	$1970.02
Electronics	Aug	$1986.28
Electronics	Sep	$1858.76

• List the most ordered Item (across all customers) for each month of the year 2017 with the total ordered qty **(10 pts)**

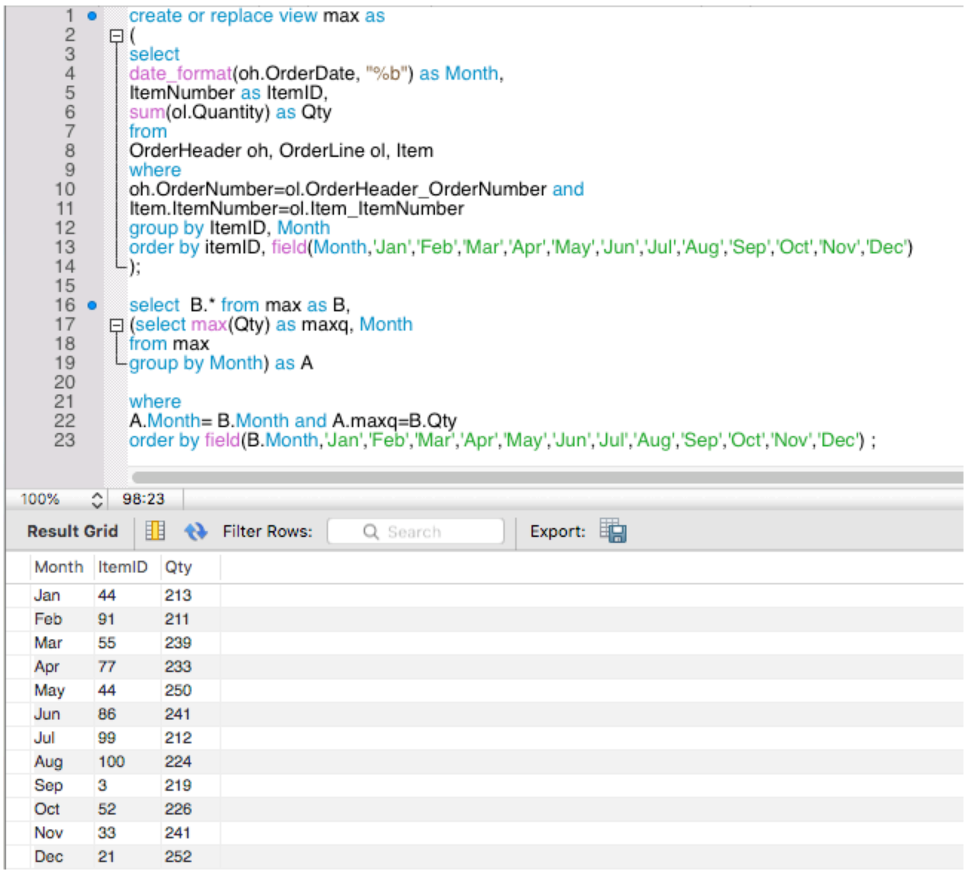
Q4.2.1 SQL scripts:

```
create or replace view max as
(
select
date_format(oh.OrderDate, "%b") as Month,
ItemNumber as ItemID,
sum(ol.Quantity) as Qty
from
OrderHeader oh, OrderLine ol, Item
where
oh.OrderNumber=ol.OrderHeader_OrderNumber and
Item.ItemNumber=ol.Item_ItemNumber
group by ItemID, Month
order by itemID, field(Month,'Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec')
);

select B.* from max as B,
(select max(Qty) as maxq, Month
from max
group by Month) as A

where
A.Month= B.Month and A.maxq=B.Qty
order by field(B.Month,'Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec')
```

Q4.1.2 SQL results:



```
create or replace view max as
(
select
date_format(oh.OrderDate, "%b") as Month,
ItemNumber as ItemID,
sum(ol.Quantity) as Qty
from
OrderHeader oh, OrderLine ol, Item
where
oh.OrderNumber=ol.OrderHeader_OrderNumber and
Item.ItemNumber=ol.Item_ItemNumber
group by ItemID, Month
order by itemID, field(Month,'Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec')
);

select  B.* from max as B,
(select max(Qty) as maxq, Month
from max
group by Month) as A

where
A.Month= B.Month and A.maxq=B.Qty
order by field(B.Month,'Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec') ;
```

Month	ItemID	Qty
Jan	44	213
Feb	91	211
Mar	55	239
Apr	77	233
May	44	250
Jun	86	241
Jul	99	212
Aug	100	224
Sep	3	219
Oct	52	226
Nov	33	241
Dec	21	252

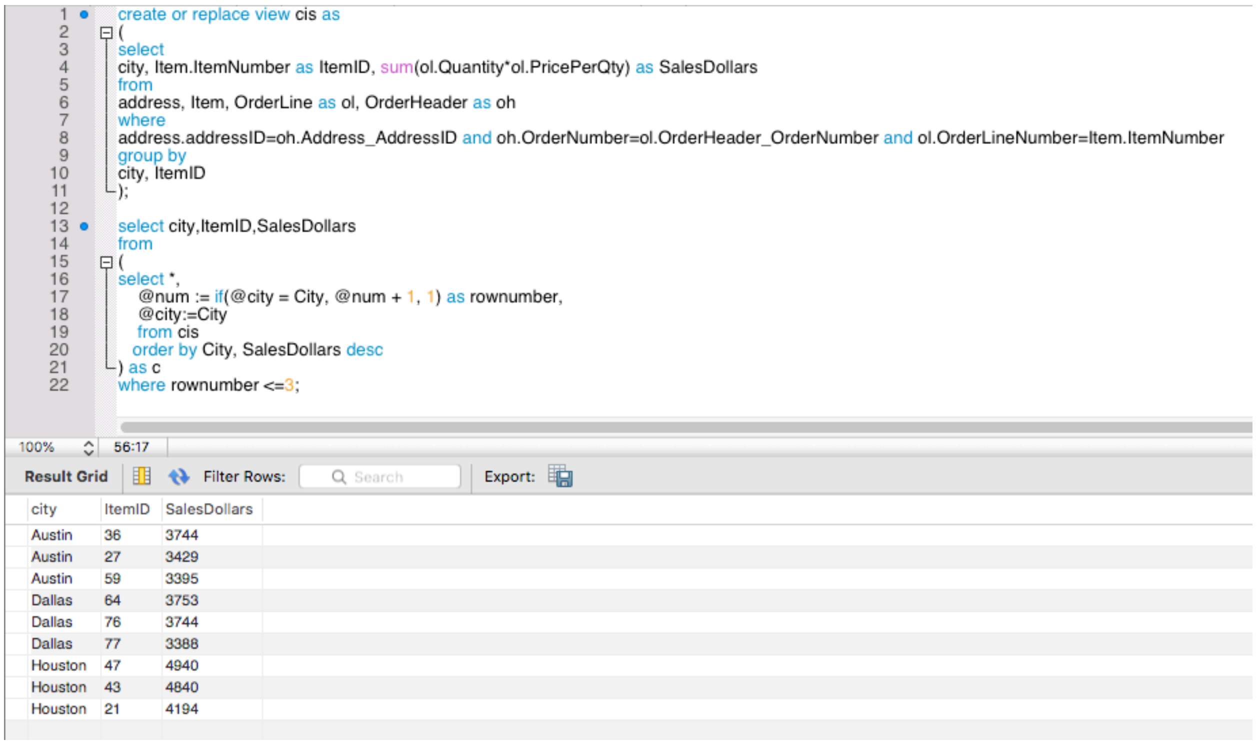
- List the top 3 items with the highest (total) sales revenue across each of the cities **(20 pts)**

Q4.3.1 SQL scripts:

```
create or replace view cis as
(
select
city, Item.ItemNumber as ItemID, sum(ol.Quantity*ol.PricePerQty) as SalesDollars
from
address, Item, OrderLine as ol, OrderHeader as oh
where
address.addressID=oh.Address_AddressID and oh.OrderNumber=ol.OrderHeader_OrderNumber and
ol.OrderLineNumber=Item.ItemNumber
group by
city, ItemID
);

select city, ItemID, SalesDollars
from
(
select *,
   @num:= if(@city = City, @num + 1, 1) as rownumber,
   @city:=City
   from cis
  order by City, SalesDollars desc
) as c
where rownumber <=3;
```

Q4.1.2 SQL results:



```
create or replace view cis as
(
select
city, Item.ItemNumber as ItemID, sum(ol.Quantity*ol.PricePerQty) as SalesDollars
from
address, Item, OrderLine as ol, OrderHeader as oh
where
address.addressID=oh.Address_AddressID and oh.OrderNumber=ol.OrderHeader_OrderNumber and ol.OrderLineNumber=Item.ItemNumber
group by
city, ItemID
);

select city,ItemID,SalesDollars
from
(
select *,
   @num := if(@city = City, @num + 1, 1) as rownumber,
   @city:=City
   from cis
  order by City, SalesDollars desc
) as c
where rownumber <=3;
```

```
city     ItemID  SalesDollars
Austin   36      3744
Austin   27      3429
Austin   59      3395
Dallas   64      3753
Dallas   76      3744
Dallas   77      3388
Houston  47      4940
Houston  43      4840
Houston  21      4194
```