Database Management Systems:

Assignment 3

Albert Adamson

Department of Computer Science. Saint Joseph's University

CSC 351: Database Management Systems

Dr. Forouraghi

October 21, 2025

Problem 1: List names and sellers of products that are no longer available (quantity=0)

Query + Results:

```
112   -- 1. List names and sellers of products that are no longer available (quantity=0)
113 • select p.name as Product_name, m.name as Seller_name
114   from merchants as m                -- Used for seller name
115   inner join sell using(mid)         -- Has quantity variable
116   inner join products as p using(pid) -- Used for product name
117   where sell.quantity_available = 0;  -- Check that quantity = 0
118
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	Product_name	Seller_name
1	Router	Acer
2	Network Card	Acer
3	Printer	Apple
4	Router	Apple
5	Router	HP
6	Super Drive	HP
7	Laptop	HP
8	Router	Dell
9	Ethernet Adapter	Lenovo

In this query, I first listed the name from the product name (aliased as Product_name) and the name from the merchant's table (aliased as Seller_name). I then joined the merchants, sell, and products tables. Since I needed data from the products table and the merchants table, the only way to combine the tables was to use the sell table. This is because the sell table has both the mid and pid as foreign keys. Once I combined everything I checked which items have a quantity_available of 0.

Problem 2: List names and descriptions of products that are not sold.

Query + Results:

```
120   -- 2. List names and descriptions of products that are not sold.
121 • select p.name, p.description    -- Only need name and description
122   from products as p              -- Products contains the all the info needed
123   where p.pid not in (select pid  -- Subquery gives the pid of every item that has been sold
124                        from sell);
125
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	name	description
1	Super Drive	External CD/DVD/RW
2	Super Drive	UInternal CD/DVD/RW

In this query, I first selected the product name and the product description since this was the only information I needed to output. I then created a subquery that listed every product that is actively being sold by a company, which is every pid inside of the sell table. I then listed every pid which was not inside of that table.

Problem 3: How many customers bought SATA drives but not any routers?

Query + Results:

```
127   -- 3. How many customers bought SATA drives but not any routers?
128 • select count(distinct c.cid) as number_customers          -- Just need a count
129   from customers as c                                       -- Only compare cid in the subqueries
130   where c.cid in (    select p.cid                          -- Subquery finds every cid of someone
131                       from place as p                       -- who has bought a SATA Drive
132                       inner join contain using(oid)
133                       inner join products as pr using(pid)
134                       where pr.description like '%SATA%')   -- SATA is in the Description not name
135   and c.cid not in (  select p2.cid                         -- Subquery finds every cid of someone (Use not in from main query)
136                       from place as p2                      -- who has bought a Router
137                       inner join contain using(oid)
138                       inner join products as pr2 using(pid)
139                       where pr2.category = 'Router');       -- Router is a category unlike SATA Drive
140
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	number_customer
1	20

In this query, I have “count(distinct c.cid)” as the only selection since that is what the question asks for. Also the distinct c.cid makes sure that the same person isn’t counted twice. Furthermore, I have two subqueries. The first one lists the customer id of everyone who has purchased an item with the word “SATA” inside of the description. This is because there isn’t a

dedicated category for SATA drives so I had to pull that information from the description. The next subquery does a similar thing, just rather than finding "SATA" in the description it finds a product with "Router" as its category since Routers are a dedicated category. Going back to the main query, a cid only gets counted if it can be found within the SATA subquery but not found in the other subquery.

Problem 4: HP has a 20% sale on all its Networking products.

Query + Results:

```
142   -- 4. HP has a 20% sale on all its Networking products.
143 • select p.pid, p.name as product_name, s.price as old_price, s.price * 0.8 as new_price  -- Show discounted price
144   from merchants as m
145   inner join sell as s on s.mid = m.mid
146   inner join products as p on s.pid = p.pid
147   where p.category = "Networking" and m.name = "HP";  -- The sale is only on HP Networking products
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	pid	product_name	old_price	new_price
1	8	Router	1034.46	827.568
2	10	Network Card	1154.68	923.744
3	12	Network Card	345.01	276.008
4	13	Network Card	262.20	209.760
5	16	Ethernet Adapter	1260.45	1008.360
6	18	Router	205.56	164.448
7	19	Router	1474.87	1179.896
8	20	Router	552.02	441.616
9	23	Router	100.95	80.760
10	28	Network Card	1179.01	943.208

In this subquery, I interpreted it as taking the original values and saying what the discount would be if it was 20%. In the selection, I multiplied the price by 0.8 to show what the discount would be. I then joined the merchants, sell, and products table since I needed information from all of the tables. Lastly, in the "where" statement I made sure to filter only HP merchant items that are also Networking products.

Problem 5: What did Uriel Whitney order ~~from Acer~~? (make sure to at least retrieve product names and prices).

Query + Results:

```
149   -- 5. What did Uriel Whitney order from Acer? (make sure to at least retrieve product names and prices).
150 • select p.name as product_name, p.description, sell.price, count(*) as quantity
151   from merchants as m
152   inner join sell using(mid)
153   inner join products as p using(pid)
154   inner join contain using(pid)                          -- Assume that each PID is sold by one Merchant (Impossible otherwise)
155   inner join place using(oid)
156   inner join customers as c using(cid)
157   where c.fullname = 'Uriel Whitney' and m.name = 'Acer'  -- Check for entries Acer and Uriel Whitney
158   group by p.pid                                         -- Apply the count aggregate to the number of times a product appears
159   order by p.name;                                       -- Unneeded I just like how it looks
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	product_name	description	price	quantity
1	Desktop	Intel Core i7-2630Q…	311.06	2
2	Ethernet Adapter	High Performance …	446.62	3
3	Hard Drive	500GB Red-Hot Ski…	1151.28	3
4	Hard Drive	640GB USB 2.0 Port…	836.99	4
5	Hard Drive	2TB Internal SATA	333.71	2
6	Laptop	Core i7 /17.3 / 750/8…	33.50	4
7	Laptop	1.66GHz Processor …	522.73	1
8	Laptop	Intel Core i5-2410M …	247.96	5
9	Monitor	LED 22-inch Backlit	1103.47	4
10	Monitor	27-inch LED	1435.38	1
11	Network Card	24 Port Gigabit Rack…	405.40	3
12	Network Card	MegaPlug AV 200 Mbs	130.43	4
13	Network Card	Wireless a/b/g/n	837.12	6
14	Network Card	Livewire Powerline A…	609.20	3
15	Printer	Black & White Laser …	836.28	4
16	Printer	Color Laser	1345.37	4
17	Printer	All-in-one	310.83	4
18	Router	Wireless N HD Medi…	945.51	1
19	Router	Wireless Dual Band …	780.65	3
20	Router	54 Mbps 4-port Wire…	394.04	3
21	Router	Wireless-G Broadba…	1256.57	4
22	Router	Gigabit Router with …	521.07	3
23	Super Drive	External CD/DVD R…	1124.26	1
24	Super Drive	DVD/CD/RW IDE	1015.95	5
25	Super Drive	DVD+R 8X DVD+R…	1135.30	2
26	Super Drive	USB 2.0 Slot-Loadin…	671.75	4
27	Super Drive	12x Internal Blu-ray …	356.13	6

Problem 5 is the first problem that I had to make an assumption about. I assumed that each product (pid) could only be sold by one Merchant. I made this assumption because this problem would be impossible without it, since the table “contain” doesn’t state who the customer bought it from, just that they bought the item. This causes issues because if a product is sold by two different merchants, then it would get counted twice if that product id is found inside of the table “contain”. After making that assumption, I listed all columns that I needed, and joined the respective tables. I then applied the “where” filter to assure that all products were from ‘Uriel Whitney’ and ‘Acer’. I then grouped by the product id, so I could apply the count() aggregate

from the selection statement to the resulting group. Additionally, the ordering I did at the end had nothing to do with the problem, I just liked the way that it looked.

Problem 6: List the annual total sales for each company (sort the results along the company and the year attributes).

Query + Results:

```
-- 6. List the annual total sales for each company (sort the results along the company and the year attributes).
select m.name as company_name, YEAR(pl.order_date) as year, sum(sell.price) as total_sales  -- Use YEAR() function to get only the year from the datetime
from merchants as m
inner join sell using(mid)
inner join contain using(pid)                          -- Assume that each PID is sold by one Merchant (Impossible otherwise)
inner join place as pl using(oid)
group by m.name, year                                                                        -- Group by name, year so the sum() aggregate is applied only to individual company yearly totals
order by m.name, year desc;                                                                  -- Sort by name than the year
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

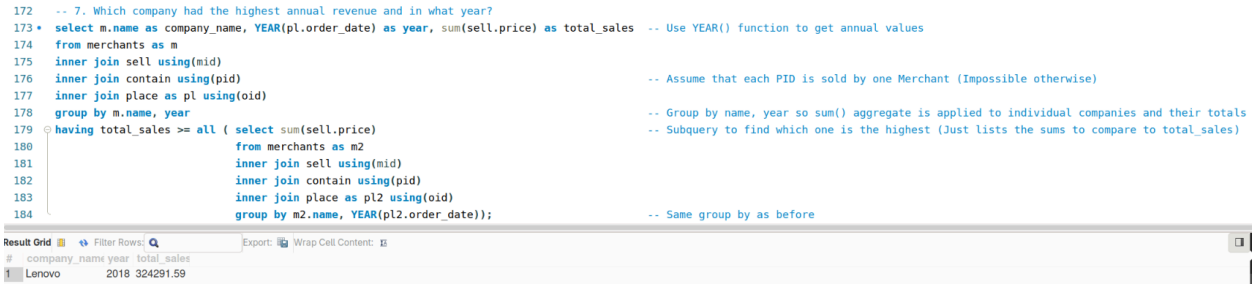
#	company_name	year	total_sales
1	Acer	2020	182311.15
2	Acer	2019	208815.80
3	Acer	2018	262059.29
4	Acer	2017	176722.77
5	Acer	2016	60291.14
6	Acer	2011	152986.30
7	Apple	2020	216461.06
8	Apple	2019	231573.17
9	Apple	2018	300413.23
10	Apple	2017	179560.78
11	Apple	2016	64748.46
12	Apple	2011	166822.91
13	Dell	2020	208063.08
14	Dell	2019	221391.83
15	Dell	2018	315004.82
16	Dell	2017	182288.61
17	Dell	2016	71462.87
18	Dell	2011	181730.35
19	HP	2020	180775.18
20	HP	2019	173334.01
21	HP	2018	222707.08
22	HP	2017	136092.43
23	HP	2016	56986.12
24	HP	2011	141030.15
25	Lenovo	2020	214154.25
26	Lenovo	2019	232610.80
27	Lenovo	2018	324291.59
28	Lenovo	2017	197980.33
29	Lenovo	2016	70131.57
30	Lenovo	2011	184939.41

In this problem, I made the same assumptions that I had made about the previous problem. In the selection statement I added an additional function to apply on top of the date, the YEAR() function. This function was used to extract just the date from the datetime which is in ‘YYYY-MM-DD’ format. This isn’t an aggregate function and can be used without some sort of grouping like sum() needs. I then joined the respective tables, and then grouped by merchant name and by year. This allowed me to sort data into groups of company and year, then applying

the sum() aggregation on top of the grouped data. Lastly, I ordered by merchant name and year descending.

Problem 7: Which company had the highest annual revenue and in what year?

Query + Results:



```
-- 7. Which company had the highest annual revenue and in what year?
select m.name as company_name, YEAR(pl.order_date) as year, sum(sell.price) as total_sales  -- Use YEAR() function to get annual values
from merchants as m
inner join sell using(mid)
inner join contain using(pid)                                                               -- Assume that each PID is sold by one Merchant (Impossible otherwise)
inner join place as pl using(oid)
group by m.name, year                                                                       -- Group by name, year so sum() aggregate is applied to individual companies and their totals
having total_sales >= all ( select sum(sell.price)                                          -- Subquery to find which one is the highest (Just lists the sums to compare to total_sales)
                            from merchants as m2
                            inner join sell using(mid)
                            inner join contain using(pid)
                            inner join place as pl2 using(oid)
                            group by m2.name, YEAR(pl2.order_date));                        -- Same group by as before
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	company_name	year	total_sales
1	Lenovo	2018	324291.59

In Problem 7, I made the same assumptions that I did in both 5 and 6. In this query, I used the similar attributes as question 6, with the main difference this time being the inclusion of the “having” statement. The “having” statement was used to pull the group that had the highest annual revenue. This works by including a subquery that lists just all of the revenues of each (company, year) pair. I then select the value that is greater than or equal to all of the revenues. This would result in me just having the company with the highest annual revenue and the year it was.

Problem 8: On average, what was the cheapest shipping method used ever?

Query + Results:

```
186   -- 8. On average, what was the cheapest shipping method used ever?
187 • select shipping_method, avg(shipping_cost)                    -- Interpreted as which shipping method is usually the cheapest
188   from orders
189   group by shipping_method                                     -- Group by the shipping methods
190   having avg(shipping_cost) <= all (  select avg(shipping_cost)  -- We want the smallest value (Subquery finds all averages per shipping method)
191                                       from orders
192                                       group by shipping_method);  -- Apply avg() aggregate to the grouped shipping methods
193
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	shipping_metho	avg(shipping_cos
1	USPS	7.455761

In this query, I interpreted it as asking which shipping method is usually cheaper on average. As a result, I only pulled the shipping method name and the average shipping cost. I also only needed to pull from the table "orders" since it contains all of the information that I needed. I also grouped by the shipping method since I wanted to apply the average aggregate to each group. Once this was done, I used a having statement to pull the group which had the least average shipping cost by using a subquery which listed each groups' average shipping cost and finding the group which was less than or equal to all the other groups.

Problem 9: What is the best sold ($) category for each company?

Query + Results:

```
194   -- 9. What is the best sold ($) category for each company?
195 • select m.name as company_name, p.category, sum(sell.price) as revenue   --
196   from merchants as m
197   inner join sell using(mid)
198   inner join products as p using(pid)
199   inner join contain using(pid)                                  -- Assume that each PID is sold by one Merchant (Impossible otherwise)
200   group by m.name, p.category                                    -- Group by company, category to apply the sum aggrate to them
201   having sum(sell.price) >= all ( select sum(sell.price)         -- Subquery is used to find the best for each company
202                                   from merchants as m2
203                                   inner join sell using(mid)
204                                   inner join products as p2 using(pid)
205                                   inner join contain using(pid)
206                                   where m.mid = m2.mid           -- m.mid comes from main query (So we only compare similar companies)
207                                   group by m.name, p2.category);  -- Group by same values as the main query
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	company_name	category	revenue
1	Acer	Peripheral	751705.66
2	Apple	Peripheral	725401.44
3	HP	Networking	446802.87
4	Dell	Peripheral	690326.49
5	Lenovo	Peripheral	702791.94

In Problem 9, I had to make a similar assumption as problems 5, 6, and 7. In the query, I needed the merchant name, product category, and a sum for each thing sold in the category. After pulling from the respective tables, I grouped by the merchant name and the product category, so I could apply the sum aggregate to the values inside of the groups. I then added a having statement with a subquery that found the best product category for each company. The way I did this was inside of the subquery's "where" clause I made sure to add the merchant id from the main query. This would make sure that each company is only compared to itself, for example, Acer's category revenue wouldn't be compared to Apple's. Within the main query's "having" clause I then made sure that the total revenue was greater than every value inside of the subquery, giving me the product category which made the most money for each company.

Problem 10: For each company find out which customers have spent the most and the least amounts.

Query + Results:

```
194   -- 9. What is the best sold ($) category for each company?
195 • select m.name as company_name, p.category, sum(sell.price) as revenue   --
196   from merchants as m
197   inner join sell using(mid)
198   inner join products as p using(pid)
199   inner join contain using(pid)                                    -- Assume that each PID is sold by one Merchant (Impossible otherwise)
200   group by m.name, p.category                                      -- Group by company, category to apply the sum aggrate to them
201   having sum(sell.price) >= all ( select sum(sell.price)           -- Subquery is used to find the best for each company
202                                   from merchants as m2
203                                   inner join sell using(mid)
204                                   inner join products as p2 using(pid)
205                                   inner join contain using(pid)
206                                   where m.mid = m2.mid             -- m.mid comes from main query (So we only compare similar companies)
207                                   group by m2.name, p2.category);  -- Group by same values as the main query
208
209
210   -- 10. For each company find out which customers have spent the most and the least amounts.
211 • select m.name as company_name, c.fullname as customer_name, sum(sell.price) as total_spent
212   from merchants as m
213   inner join sell using(mid)
214   inner join contain using(pid)                                 -- Assume that each PID is sold by one Merchant (Impossible otherwise)
215   inner join place using(oid)
216   inner join customers as c using(cid)
217   group by m.mid, c.cid
218   having sum(sell.price) >= all ( select sum(sell.price)        -- Calculates which customer spent the most
```

```
218  having sum(sell.price) >= all ( select sum(sell.price)            -- Calculates which customer spent the most
219                                  from merchants as m2
220                                  inner join sell using(mid)
221                                  inner join contain using(pid)
222                                  inner join place using(oid)
223                                  inner join customers as c2 using(cid)
224                                  where m.name = m2.name             -- Verify that we are comparing the same company as main query
225                                  group by m2.mid, c2.cid)           -- Groups by the same as the main query
226  or sum(sell.price) <= all ( select sum(sell.price)                 -- Calculates which customer spent the least amount (Joins it to output by an or)
227                              from merchants as m3
228                              inner join sell using(mid)
229                              inner join contain using(pid)
230                              inner join place using(oid)
231                              inner join customers as c3 using(cid)
232                              where m.name = m3.name                 -- Verify that we are comparing the same company as main query
233                              group by m3.mid, c3.cid)               -- Group by the same as the main query
234  order by m.name, sum(sell.price) desc;                             -- Not needed, I just like the way that it looks
235
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	company_name	customer_name	total_spen
1	Acer	Dean Heath	75230.29
2	Acer	Inez Long	31901.02
3	Apple	Clementine Travis	84551.11
4	Apple	Inez Long	32251.10
5	Dell	Clementine Travis	85611.55
6	Dell	Inez Long	31135.74
7	HP	Clementine Travis	66628.06
8	HP	Inez Long	26062.89
9	Lenovo	Haviva Stewart	83030.26
10	Lenovo	Inez Long	33948.91

Within Problem 10, I made the same assumption as I did with problems 5, 6, 7, and 9, since I was also using the table “contain”. In this problem’s query, I pull the merchant name, customer full name, and the sum of each customer’s order costs. I then pulled from the respective queries and grouped by the merchant id and the customer id to create groups of (merchant, customer) and apply the sum aggregate to these groups. Within the having statement I have two subqueries. The first subquery finds which of the customers spent the most at that company. In this subquery, I made sure to place a where clause within that made sure to only pull the same merchant id. This would allow each statement to only compare the sums of the customers who bought things from the same company (Similar problem as problem 9). After the first subquery executes, I made sure that the current sum is greater than or equal to all of the values to find the customer with the highest total spent at that company. The second subquery does the opposite and finds the customer who spent the least at that company. It’s important to note that the two subqueries are practically the same, the only difference is the first one is greater than or equal to, and the second

one is less than or equal to. The two queries are joined by an “or”, so both the person who spent the least and the person who spent the most are added to the same main query result.

Appendix:

ER Diagram: