Cheat Sheet Final Exam 1

- Write a SQL Query that returns the total number of customers from city = 'Nyc'

→ select count (*) from customers where city = 'Nyc';

- Write a SQL query that returns the number of NOT NULL price each values in the order details table where the product code = 'S24-3969'

→ select count (price each) from orderdetails where product code = 'S24-3969';

for distinct → count (distinct price each)

- Write a SQL Query that returns the product code and sum of the quantity ordered of all orderdetails table where the productcode = 'S24-2840'

→ select product code, sum (quantity ordered) as total_quantity from orderdetails where product code = 'S24-2840';

- Aggregate functions queries

→ select sum (quantity ordered)

→ select avg (price each) — weighted average

→ select avg (distinct priceeach) — unweighted average; aggregate + distinct inside bracket

→ select variance (priceeach)

→ select min (priceeach) as min_price, max (price each) as max

- Write a SQL Query that returns all product code with atleast 25 outstanding orders in the orderdetails table

→ select product code from orderdetails group by productcode having count (*) >= 25;

- Write a SQL Query that returns all productcode and total quantity ordered with the total quantity ordered exceeds 1000 in the orderdetails table

→ select productcode from orderdetails group by product code having sum (quantity ordered) > 1000.

- Write a SQL query that returns a list of outstanding orders, ordered ascending by date and desc by cust. number.

→ select ordernumber from orders order by orderdate asc, customernumber desc;

- Write a SQL Query that returns product code, ordernumber, priceeach with product code = 'S24-2840' order by 3rd column in descending order

→ select product code, ordernumber, priceeach from orderdetails where productcode = 'S24-2840' order by 3 desc;

- Write a SQL query that returns for each customer with outstanding orders, customernumber, customername, together with the ordernumber, date, the productcode, productname and quantity ordered.

→ select c.customernumber, c.customername, o.ordernumber, o.orderdate, p.productcode, p.productname, od.quantityordered from customers c, orders o, orderdetails od, products p where c.customernumber = o.customernumber and o.ordernumber = od.ordernumber and od productcode = p.productcode

- Write a SQL query that returns all pairs of customers who are located in the same city

→ select c1 customername, c2. customername, c1. city, from customers c1, customers c2 where c1. city = c2. city and c1. customer number < c2. customernum

- Write a SQL Query that returns customernumber, customername who have ordered atleast one type of productlines = 'planes'

→ select c.customernumber, c. customername from customers c, orders o, orderdetails od, products p where c.customernumber = o. customernumber and o.ordernumber = od. ordernumb ↑ od. product code = p.productcode and p. productline = 'planes';

- Write a SQL Query that returns the productcode, productname, and total quantityordered for each product that is in an order

→ select od. productcode, p. productname, sum (quantityordered) as totalquantity from orderdetails od, products p where od. productcode = p. productcode group by od. productcode;

- Write a SQL query that returns customernumber, customername, and if applicable, include the ordernumbers of their orders. Namely the information of all customer even if they do not have any order

→ select c.customernumber, c. customer name, o. ordernumber from customers as c left outer join orders as o on c. customer number = o. customernumber;

- Write a SQL query that returns all productcode, together with the productname and total quantity ordered, even if there are currently no outstanding orders for a product

→ select p. productcode, p. productname, sum (od. quantity ordered) from orderdetails as od right outer join products as p on od. product code = p. product code group by p. product code;

- Write a nested SQL query that returns the name of the customer with whom the ordernumber '10202' is placed.

→ select customername from customers where customernumber = (select customernumber from orders where ordernumber = '10202'

- Write a nested SQL query that returns the names of the customers who ordered productcode = 'S24-2840'

→ select customername from customers where customernumber in (select o.customernumber from orders o, orderdetails od where o.ordernumber = od.ordernumber and od.productcode = 'S24-2840');

- Write a nested SQL query that returns the names of the customers who ordered the product with product code = 'S24-2840' and the product with code = 'S50-1341'

→ select customername from customers where customernumber in (select o.customernumber from orders o, orderdetails od where o.ordernumber = od.ordernumber and od.productcode = 'S24-2840') and customernumber in (select o.customernumber from orders o, orderdetails od where o.ordernumber = od.ordernumber and od.productcode = 'S50-1341');

- Write a correlated SQL query that returns the productname of all products with atleast 5 orders

→ select p.prodname from products p where (select count(*) from orderdetails od where od.productcode = p.productcode group by od.productcode >= 5

- Write a correlated SQL Query that returns the customernumber, customername of the customers who ordered a product at a priceeach lower than the average priceeach of that product, together with the productcode, productname priceeach and quantityordered

→ select c.customernumber, c.customername, p.productcode, p.productname, od.priceeach, od.quantityordered from customers c, products p, orders o, orderdetails od where c.customernumber = o.customernumber and o.ordernumber = od.ordernumber and od.productcode = p.productcode and od.priceeach < (select avg(priceeach) from orderdetails where productcode = p.productcode);

- Write a correlated SQL Query that returns the customernumber, customername with the top 2 most orders

→ select customernumber, customername from customers where customernumber in (select m1.customernumber from (select customernumber, count(*) as num from orders group by customernumber) as m1 where 2 > (select count(*) as num from orders group by customernumber) as m2 where m1.num < m2.num));

- Write a SQL query that returns the customernumber, customername who ordered the productcode = 'S18-3136' with the highest priceeach

→ select c.customernumber, c.customername from customers c where c.customernumber = (select o.customernumber from orders o, orderdetails od where o.ordernumber = od.ordernumber and productcode = 'S18-3136' and od.priceeach >= all (select priceeach from orderdetails where productcode = 'S18-3136'));

- Write a SQL Query that returns the customer number, customername who ordered the productcode 'S18-3136' and did not pay the lowest priceeach.

→ select c.customernumber, c.customername from customers c where c.customernumber in (select o.customernumber from orders o, orderdetails od where o.ordernumber = od.ordernumber and od.priceeach > any(select priceeach from orderdetails where productcode = 'S18-3136'));

- Write a SQL Query using 'EXISTS' that returns the customernumber, customername who ordered the productcode 'S18-3136'

→ select c.customernumber, c.customername from customers c where exists (select * from orders o, orderdetails od where o.ordernumber = od.ordernumber and o.customernumber = od.customernumber and productcode = 'S18-3136');

- Write a SQL Query that returns the customer number, customername who are either located in 'Boston' or have ordered the productcode 'S18-3136'

→ select customernumber, customername from customers where city = 'Boston' union select c.customernumber, c.customername from customers c, orders o, orderdetails od where c.customernumber = o.customernumber and o.ordernumber = od.ordernumber and od.productcode = 'S18-3136';

- Write a SQL Query that returns customer number, customer name who have not ordered any product

[not using EXCEPT as not supported by MySQL]

→ Select customernumber, customername from customers where customernumber not in (select customernumber from orders);

● Write a SQL Query that returns the customer number, customername who are located in 'Boston' and have ordered the product code 'S18-3136' [not using INTERSECT as not supported by MySQL]

→ Select c.customernumber, c.customername from customers c, orders o, orderdetails od where c.customernumber = o.customernumber and o.ordernumber = od.ordernumber and c.city = 'Boston' and od.productcode = 'S18-3136';

→ CLASSIC MODELS SCHEMA

① CUSTOMERS : customernumber, customername, contactlastname, contactfirstname, phone, addressline 1, addressline 2, city, state, postal code, country, salesRepemployeenumber, creditlimit

② EMPLOYEES : employeenumber, lastname, firstname, extension, email, officecode, reportsTo, jobtitle

③ OFFICES: officecode, city, phone, addressline 1, addressline 2, state, country, postalcode, territory

④ ORDERDETAILS: ordernumber, productcode, quantityordered, priceeach, orderlinenumber

⑤ ORDERS: ordernumber, orderdate, requireddate, shippeddate, status, comments, customernumber

⑥ PAYMENTS: customernumber, checknumber, paymentdate, amount

⑦ PRODUCTLINES: productline, textdescription, htmldescription, image

⑧ PRODUCTS : productcode, productname, productline, productscale, productvendor, quantitydescription, quantityinstock, buyprice, MSRP

→ CYPHER QUERIES

① MATCH (me:Reader) -- (friend:Reader) -- (b:Book)
WHERE g.name = 'Romance' AND me.name = 'Bart Baesen'
RETURN DISTINCT friend.name

Q. Who are Bart's friends that liked Romance books?

② Can you recommend humor books that Seppe's friend liked and Seppe has not liked?

↳ MATCH (me:Reader) -- (friend:Reader), (friend) -- (b:Book), (b) -- (g:genre)
WHERE NOT (me) -- (b)
AND me.name = 'Seppe vanden Broucke'
AND g.name = 'humor'
RETURN DISTINCT b.title

③ Can you list all pairs of friends that like no books in common?

↳ MATCH (b1:Book), (b2:Book)
WITH b1, b2
OPTIONAL MATCH (b1) -- (g.genre) -- (b2)
WHERE g is NULL
RETURN b1.title, b2.title

④ Query returns a list of distinct customer names who purchased a book written by Wilfried Lemahieu are older than 30 and paid in cash

↳ MATCH (c:CUSTOMER) -[p:PURCHASED]-> (b:Book) <-[:WRITTEN BY]- (a:AUTHOR) WHERE a.name = "Wilfried...."
AND c.age >30 AND p.type = "cash"
RETURN DISTINCT c.name

⑤ MATCH (b1:Book) -- (g:genre) -- (b2:Book)
WITH b1, b2, count(*) AS common_humor
WHERE g.name = "Humor" AND common_humor >= 2
RETURN b1.title, b2.title

↳ Q. Can you list the pair of friends that like atleast 2 humor books in common

→ SQL QUERIES ORDER

① Select
② From
③ Where
④ Group By
⑤ Having
⑥ Order By

● Query using a map-reduce to get count of restaurant for each location

→ var mapFunction = function() {emit(this.location, {count:1});};
var redFunction = function(location, val) {var value = {count:0};
for(i=0; i<val.length; i++){value.count += val[i].count;}
return value;}, db.restaurants.mapReduce(mapFunction, redFunction, {out:'count_of_restaurants_for_each_location'});
db.count_of_restaurants_for_each_location.find().pretty();

● RECURSIVE QUERY

WITH SUBORDINATES (SSN, NAME, SALARY, MNGR, LEVEL) AS
(SELECT SSN, NAME, SALARY, MNGR, 1 FROM EMPLOYEE
WHERE MNGR = NULL)
UNION ALL
(SELECT E.SSN, E.NAME, E.SALARY, E.MNGR, S.LEVEL + 1
FROM SUBORDINATES AS S.EMPLOYEE AS E
WHERE S.SSN = E.MNGR)
SELECT * FROM SUBORDINATES
ORDER BY LEVEL

● Write a query using a map-reduce pipeline to get the highest (maximum) rating for each restaurant_id

→ var mapFunction = function() {emit(this_id, this.rating);};
var reduceFunction = function(_id, ratingval) {return Array.max(ratingval);};
db.restaurants.mapReduce(mapFunction, reducefunction, {out: "max_rating"})
db.Max_rating.find().pretty()

- Write a query to get the average rating for each restaurant_id without using a map-reduce pipeline.

→ db.restaurants.aggregate([
{$project: {Avg_rating: {$avg: "$rating"}}}]);

- Write a query to get the average rating for each location without using a map-reduce pipeline.

→ db.restaurants.aggregate([
{$group: {_id: '$location', avg: {$avg: '$rating'}}},
{$sort: {_id: 1}}]);
→ 1 – ascending
-1 – descending

- Find all friends that have more than one book in common.

→ MATCH (a1:Reader) -- (b:Book) --(a2:Reader)
WITH a1, a2, count(*) as common_books
WHERE (a1) -- (a2) AND common_books > 1
RETURN a1.name, a2.name, common_books;

- Find a list of all Books that Bart likes but Seppe doesn't.

→ MATCH (a1:Reader) -- (b:Book), (a2:Reader) -- (a1:Reader)
WHERE NOT (a2) -- (book) AND a1.name = 'Bart' AND
a2.name = 'Seppe' RETURN b.title;

- Find a count of all Bart's friends, but only if at least one FRIENDS_OF relation exists. Output his name & count of friends as NumberofFriends.

→ OPTIONAL MATCH (r:Reader) -[:FRIEND_OF] -(friend:
Reader) WHERE r.name = 'Bart' RETURN r.name,
count(friend) AS NumberOfFriends

- Return the genres the friends of Seppe like together with how many times they liked a book of that genre. Sort by how many times decreasing.

→ MATCH (a:Reader) -- (d:Reader) -- (b:Book) -[s: IS_GENRE] -
(genre:Genre) WHERE a.name = 'Seppe' RETURN genre.name,
COUNT(s) ORDER BY COUNT(s) DESC;

- Write a query to get the type of food with highest average rating without using a map-reduce pipeline.

→ db.restaurants.aggregate([{$group: {_id: "$type_of_food",
Max_Avg_Rating: {$avg: "$rating"}}}, {$sort: {
Max_Avg_Rating: -1}}, {$limit: 4}])

→ Previous Year Question Papers Answers:

- Select D.name From DEPT D
Join IN_DEPT ID on ID.DID = D.DID
Join EMP E on E.EID = ID.EID
where E.salary = (select min(salary) from EMP);

- Select B.BID, B.Bname from BUILDING B
Join IN_BUILDING IB ON IB.BID = B.BID
Group by IB.BID Having count(IB.EID) > 50;

- Select D.name From DEPT D
join IN_DEPT ID on ID.DID = D.DID
group by ID.DID having count(ID.EID) is null;

- Select E.EID, E.name from EMP E join IN_DEPT ID
on ID.EID = E.EID join DEPT D on D.DID = ID.DID
Where D.Annual Budget in
(Select AnnualBudget from DEPT order by Annual Budget
Desc limit 3);

- Select D.DID, D.Dname, count(E.EID) AS Total_emp,
avg(E.salary) AS avg_sal, (D.AnnualBudget/count(E.EID)) AS
AnnualBudgetperEmp From DEPT D join IN_DEPT ID
on ID.DID = D.DID join EMP E on E.EID = ID.EID
group by D.DID having count(E.EID) >= 1;

- Write a recursive SQL query to list all prereq courses for the course "Principles of Database Mgmt"

→ WITH RECURSIVE PREREQ(coursenr, pre_req_coursenr, level)
AS(SELECT coursenr, pre_req_coursenr, 1 FROM
pre_requisite WHERE coursenr = (SELECT coursenr FROM
coursename = "Principles of Database Management")
UNION ALL
SELECT C.coursenr, C.pre_req_coursenr, P.level + 1
FROM PREREQ AS P, pre_requisite AS C
WHERE P.pre_req_coursenr = C.coursenr)
SELECT P.coursenr, C1.coursename, P.pre_req_coursenr,
C2.coursename, P.level FROM PREREQ P, course C1,
course C2 WHERE P.coursenr = C1.coursenr AND
P.pre_req_coursenr = C2.coursenr ORDER BY P.level