**COP4710 – Theory and Structure of Databases**

**Summer 2016**

**Homework 5**

Due Sunday Night, June 12, 2016

***Eric Adams***

***MySQL at FSU:***

Use your account at the ***CS Department’s MySQL server***. For the first six questions, upload the ***cia.sql*** file (attached to the assignment) to your CS account. Refer to the Read-Me 2030 in the Hands-On section to see how to do this and what's in a .sql file. (Windows users have to install an application to upload to Unix.)…

…OR… You can copy cia.sql from your instructor's account. From your *CS command line*, this command will copy the file to your home directory:

* **cp /home/faculty/lockwood/cop4710public/cia.sql ~**

Then, log into MySQL and run this command from your **mysql>** prompt:

* **source cia.sql**

The cia.sql file will create and populate the **cia** table, which has this schema:

**cia ( name , region , area , population , gdp )**

*Factoid:*

This table contains (outdated) statistics about the world's countries, and yes, the CIA did indeed publish this data: <https://www.cia.gov/library/publications/the-world-factbook/>

**Aggregate Functions** (These do not require GROUP BY clauses).

1. Give the total gdp of the region 'North America'. (The query result is 10844874000000)

SELECT SUM(gdp)

FROM cia

WHERE region = ‘North America’;

2. How many countries have an area of at least three million? (The query result is 8)

SELECT COUNT(name)

FROM cia

WHERE area >= 3000000;

3. What is the total population of ('Belgium','Netherlands','Luxembourg')? (The query result is 26571132)

SELECT SUM(population)

FROM cia

WHERE name = ‘Belgium’ OR name = ‘Netherlands’ OR name = ‘Luxembourg’;

**GROUP BY Clause**

4. For each region, show the region and number of countries. (There are 15 regions; the first region is Africa, 59; the last region is World, 5)

SELECT region, COUNT(name)

FROM cia

GROUP BY region;

5. For each region show the region and number of countries with populations of at least 20 million. (There are 9 regions; the first region is Africa, 11; the last region is Southeast Asia, 7)

SELECT region, COUNT(name)

FROM cia

WHERE population >= 20000000

GROUP BY region;

6. List the regions with total populations of at least 200 million. (There are 6 regions; the first region is Africa, the last region is Southeast Asia)

SELECT region

FROM cia

GROUP BY region

HAVING SUM(population) >= 200000000

===========================================================

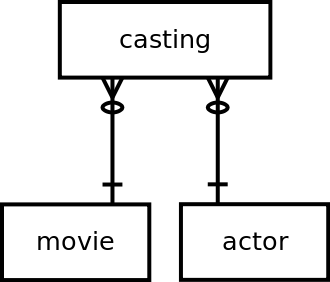
***SQLzoo:***

Use the Query Entry Box at SQLzoo.net. **Paste your *query* (but *not* the results)** below the question (just one query).

**GROUP BY and Subqueries**

Do the following problem at [**http://sqlzoo.net/wiki/More\_JOIN\_operations**](http://sqlzoo.net/wiki/More_JOIN_operations). We'll use the **Movie** database.

**ER Diagram for the Movie Database.** This database consists of a M:M relationship using an association table:



**Table schemas** (The table names and attributes):

movie ( id , title , yr , score , votes , director )

actor ( id , name )

casting ( movieid , actorid , ord )

Seeing the data helps you build a query. To see what’s in a table, run

*SELECT \* FROM movie;*

for the tables you're working with.

**GROUP BY example**

Here's a query that returns the titles of the movies and the number of actors in each:

**SELECT m.title, COUNT(c.movieid)**

**FROM movie m**

**JOIN casting c ON m.id = c.movieid**

**GROUP BY m.title**

**ORDER BY m.title;**

This problem has a couple of subtle twists:

* If you use COUNT(\*) then you get 1 instead of 0 if there are no actors cast in the movie, as is the case for “12th & Delaware”
* If you use JOIN instead of LEFT JOIN, then you don't get any row at all for a movie with no cast member.

Note the ORDER BY clause. The subquery form of the query returns rows in a different order unless you sort them, making comparison of the two queries' results difficult.

**Correlated Subquery**

7. Rewrite the above query using a corelated subquery ***instead*** of a JOIN and GROUP BY. *The subquery will go into the SELECT clause of the outer query and provide the count.* You may post hints in the discussion board.

SELECT title,

(SELECT COUNT(\*)

FROM casting

WHERE casting.movieid = movie.id)

FROM movie

ORDER BY title;

=================================================================

***MySQL at FSU Again:***

Now it's back to FSU. Upload two more files to your account (**Orders.sql**, **OrderDetails.sql**); they're attached to the assignment – Or copy them from your instructor's account as before:

* **cp /home/faculty/lockwood/cop4710public/Orders.sql ~**
* **cp /home/faculty/lockwood/cop4710public/OrderDetails.sql ~**

Run them using *SOURCE* as before from your **mysql>** prompt, and they'll build two new tables filled with rows.

* **source Orders.sql**
* **source OrderDetails.sql**

The files are part of the BIRT sample database from:  
<http://www.eclipse.org/birt/documentation/sample-database.php>

BIRT stands for *Business Intelligence and Reporting Tools*. Quoting the web site: “BIRT is an open source Eclipse-based reporting system that integrates with your Java/J2EE application to produce compelling reports”. This is under the Eclipse Public License.

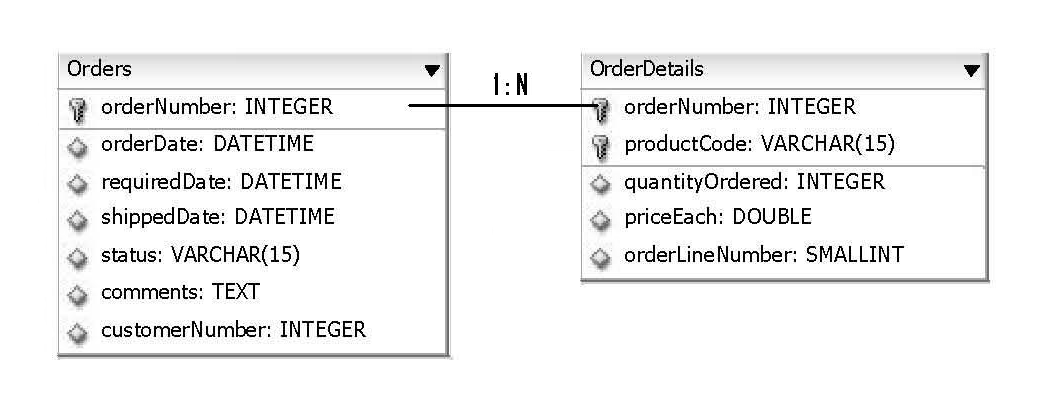
We’ll practice with part of the Line Item Pattern, the classic structure described in the *0440 Data Modeling and Implementation* Read-Me.

We’ll use only two tables:

* order headers
* order line-items (also called *details*)

These two tables have a parent/child relationship. Each *order header* row are fields from the top of an order form. Each *order line item* row is an individual line on the order form. There are several line items for each header, because an order form lets you order several items on the same form.

The OrderDetails table is ID-dependent on the Orders table. The Orders table uses a surrogate key. The key of OrderDetails is the composite (orderNumber , productCode).



You can use the **DATEDIFF(date1, date2)** function to do date math. This function subtracts date2 from date1, and returns the number of days between the two dates. The DATE data type contains both date and time. DATEDIFF is handy because it *ignores the time-of-day* component and deals *only with the date* component.

The **YEAR(date)** function returns the year component of a date.

Remember that the form of a SELECT statement is:

SELECT

FROM

JOIN ON

WHERE

GROUP BY

HAVING

ORDER BY

LIMIT

SELECT-FROM-WHERE is the basic chunk you always use. GROUP BY and HAVING go together. ORDER BY appears last. LIMIT goes after ORDER BY, but is **specific to MySQL**.

Here’s a **shortcut for joins:** if the join columns in both tables of an equijoin have the same name, then instead of writing:

ON Orders.orderNumber = OrderDetails.orderNumber

you can simplify your statement by replacing that with:

**USING**(orderNumber)

**GROUP BY Problems**

These will help you get the hang of the GROUP BY.

8. We want to see how long of a delay there is between the time an order is placed and when it ships. Write a SELECT statement to return a result containing two columns: (1) the number of days from the orderDate to the shippedDate; and (2) the number of orders falling into this category.

SELECT DATEDIFF(shippedDate, orderDate), COUNT(orderNumber)

FROM Orders

GROUP BY DATEDIFF(shippedDate, orderDate);

9. Write a SELECT statement to return the orderNumber and total amount for each order placed in 2004 that took more than 7 days from the orderDate to the shippedDate. (Hint: This doesn’t use HAVING).

SELECT orderNumber, OD.priceEach \* OD.quantityOrdered

FROM Orders

JOIN OrderDetails AS OD USING(orderNumber)

WHERE YEAR(orderDate) = 2004 AND DATEDIFF(shippedDate, orderDate) > 7

GROUP BY orderNumber;

10. Write a SELECT statement to return the orderNumber and total amount for each order totaling over $60,000.

SELECT orderNumber, SUM(priceEach \* quantityOrdered) AS Amt

FROM OrderDetails

GROUP BY orderNumber

HAVING Amt > 60000;

11. Find the orderNumber and amount for the top 5 largest orders placed. (Not just the largest line item value, but the largest dollar amount for the orders).

SELECT orderNumber, SUM(priceEach \* quantityOrdered) AS Amt

FROM OrderDetails

GROUP BY orderNumber

ORDER BY Amt DESC

LIMIT 5

Tip 1 – Use LIMIT 5 at the end of the query (the syntax is different for different DBMSs)

Tip 2 – Sorting will drive you crazy if you don't know the secret: In the SELECT clause use an alias for the sum:

SUM( … ) ***AS Amt***

Then sort by the alias:

ORDER BY ***Amt***

Remember you can sort in both ascending (default) and descending (DESC) order.