

Comp 453

Lab #1

Answers

- Look at COMPANY schema (link on syllabus)
- login to local PHPMyAdmin
 - ◆ Run MampServer
 - ◆ Open PHPMyAdmin
- Click on NEW in left menu bar
- click on SQL tab on top
 - ➔ CREATE DATABASE COMPANY;
- See the company DB in the left
- Open the CompanyDDL.txt file on your local computer (there is a link on the syllabus) (use Notepad++ or TextWranger or similar: it's a plain text file)
- Select and copy the entire file
- In PHPMyAdmin, paste into the SQL window.
Click GO
- Note that this file had both the DDL and the Insert statements!!
- Repeat the above instructions and create CompanyBackup DB
 - ◆ You can copy a DB on the command line: [mysqlDBcopy](#)
 - ◆ Alternatively, also on the command line: [mysqlDump](#)
 - ◆ -->You can also copy or rename a DB in PHPMYAdm [copy/rename](#)
- In SQL window, type SHOW TABLE STATUS; and discuss INNODB vs. MyISAM (foreign key support; transactions; row-level locking)

That was the warmup! Now for the main event.

Look at the DDL for [Pine Valley](#).

1. Display all fields in the table Customer_t.
Select * from Customer;
2. Display product name and quantity on hand from PRODUCT for all Pine Valley Furniture Company products that have a std price < \$275
SELECT ProductDescription,ProductStandardPrice
FROM Product_t
WHERE PProductStandardPrice < 275;
3. What is the address of the customer named Home Furnishings?
Use an alias, Name for the Customer Name.
SELECT CUST.CustomerName AS Name, CUST.CustomerAddress
FROM Customer_T AS Cust
WHERE Cust.CustomerName='Home Furnishings';

*NOTE: Due to the order of operations, you can't use "Name" (the alias) in the WHERE clause
But it will change the column header to Name.*

4. List the unit price, product name, and product ID for all products in the Product table.

```
SELECT ProductStandardPrice, ProductDescription, ProductID
FROM Product_t;
```

5. What are the standard price and standard price if increased by 10% for every product?

```
SELECT ProductID, ProductStandardPrice, ProductStandardPrice*1.1 AS Plus10Percent
FROM Product_t;
```

6. What is the average standard price for all products in inventory?

```
SELECT AVG(ProductStandardPrice) AS AveragePrice
FROM Product_t;
```

7. How many different items were ordered on Order number 1004?

```
SELECT COUNT(*)
FROM OrderLine_t
WHERE OrderID = 1004;
```

8. How many different items were ordered on order number 1004, and what are they?

```
SELECT ProductID, COUNT(*)
FROM OrderLine_t
WHERE OrderID=1004;
```

*Note that the result shown is 6 2. **This is NOT correct!! (scalar/aggregate)***

Look at the OrderLine table. There are two ProductIDs and a count of 2.

It retrieved the first ProductID and the count of 2.

In many versions of SQL, this would generate an error.

9. Find the difference between the std price of each product and the average price of all products:

```
SELECT ProductStandardPrice - AVG(ProductStandardPrice) FROM Product_t;
```

This results in -265.625. This is clearly incorrect. Again, mixing scalar and aggregate.

10. Display for each product the difference between its standard price and the overall average standard price of all products.

```
SELECT ProductStandardPrice - PriceAvg AS Difference
FROM Product_t, (SELECT AVG(ProductStandardPrice) AS PriceAvg FROM Product_t) AS ProdAvg;
```

This is not what we typically call a "nested query", but it acts in a similar way.

A couple of notes: MySQL requires that the "derived table" has an alias, in this case ProdAvg.

Also, the derived table is really just one value, which is "cross-product-ed" with the Product table.

This allows the FIELD name PriceAvg to be "tacked on" to each row in Product_t.

This new temporary table (Product_t cross ProdAvg) is the table against which the query is posed.

11. Alphabetically, what is the first product name in the Product table?

```
SELECT MIN(ProductDescription)
FROM Product_t;
```

12. Which orders have been placed since 10/24/2010? (date format: yyyy-mm-dd)

```
SELECT OrderID, OrderDate
FROM Order_t
WHERE OrderDate > '2010-10-24';
```

skip the entire discussion below:

*Note that our dates are not in a good format for MySQL (see DDL and Insert text file).
So let's update the dates. The format for dates is YYYY-MM-DD.*

13. UPDATE Order_t
Set OrderDate='2010-10-05';

*But this updates all of the dates. You would have to use a WHERE clause for each statement.
Better would be to DELETE all of the values for the table, and to INSERT new values.
DELETE doesn't delete the table, but only the values in the table. DROP deletes the whole table. (or DB)
DELETE from Order-t should delete all of the data. BUT there's a foreign key constraint:*

```
CONSTRAINT `OrderLine_FK1` FOREIGN KEY (`OrderID`) REFERENCES `order_t` (`OrderID`))
```

*So, you can either run a bunch of updates; or temporarily delete the foreign key constraint.
Or, just cheat and change the dates in Order-t.*

Or, drop OrderLine and Order and then recreate them and insert correct data. Options.

DDL and Insert with fixed data.

Then rerun the select query.

14. What furniture does Pine Valley carry that isn't made of cherry?

```
SELECT ProductDescription, ProductFinish
FROM Product_t
WHERE ProductFinish != 'Cherry';
```

15. Display all customers for whom we do not know their postal code.

```
SELECT * FROM Customer_T WHERE CustomerPostalCode is NULL;
```

The result is that the empty set, because all customers have a postal code.

```
UPDATE Customer_t
SET CustomerPostalcode = NULL
WHERE CustomerId = 1;
```

Or, you could just check the little box in PHPMysqlAdmin that says Null. :-)

16. List product name, finish, and unit price for all desks and tablers in the PRODUCT table that cost more than \$300.

17. Which products in the Product table have a standard price between \$200 and \$300?

```
SELECT ProductDescription, ProductStandardPrice
FROM Product_t
WHERE ProductStandardPrice BETWEEN 200 AND 300;
```

18a What order numbers are included in the OrderLine table?
Recall that OrderLine has many rows with the same OrderID since that is part of a composite key.

```
SELECT OrderID
FROM OrderLine_t;
```

18b Note duplicates.

```
SELECT DISTINCT OrderID
FROM OrderLine_t;
```

Note that DISTINCT is on every instance (that is, row):

18c List the unique combinations of order number and order qty in OrderLine:

```
SELECT DISTINCT OrderID, OrderedQuantity
FROM OrderLine_t;
```

19. List all customers who live in warmer states.

```
SELECT CustomerName, CustomerCity, CustomerState
FROM Customer_t
WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI');
```

20a. Same as previous query, but list the customers alphabetically by customer within state.

```
SELECT CustomerName, CustomerCity, CustomerState
FROM Customer_t
WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')
ORDER BY CustomerState, CustomerName;
```

Note: You could use DESC too.

20b: Same query as above, using column positions instead of names.

(Don't do this unless you have a good reason. It's still a set...)

```
SELECT CustomerName, CustomerCity, CustomerState
FROM Customer_t
WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')
ORDER BY 3, 1;
```

20c Same query, limit the number of rows in the query response.

```
SELECT CustomerName, CustomerCity, CustomerState
FROM Customer_t
WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')
ORDER BY 3, 1 LIMIT 5;
```

20d Same query, skip the first two rows of response, show the next 4.

```
SELECT CustomerName, CustomerCity, CustomerState
FROM Customer_t
WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')
ORDER BY 3, 1 LIMIT 2, 4;
```

21 Count the number of customer with addresses in each state to which we ship.

```
SELECT CustomerState, COUNT(CustomerState)
FROM Customer_t
GROUP BY CustomerState;
```

Note: 1 row in the query response corresponds to one group.

Note: If you change one of the states to NULL, it is in its own group.

22. Difference between COUNT(*) and COUNT (attribute)

Try these when one of the states has a NULL value.

23. Count the number of customers with addresses in each city to which we ship. List the cities by state.

Change "Santa Clara" to "Sacramento" and rerun the query.

24. Find only states with more than 1 customer.

```
SELECT CustomerState, COUNT(CustomerState)
FROM Customer_t
GROUP BY CustomerState
HAVING COUNT(CustomerState) > 1;
```

25. List, in alphabetical order, the product finish and the average standard price for each finish for selected finishes having an average standard price less than 750.

```
SELECT ProductFinish,AVG(ProductStandardPrice)
FROM Product_t
WHERE ProductFinish IN('Cherry','Natural Ash','Natural Maple','White Ash')
GROUP BY ProductFinish
HAVING AVG(ProductStandardPrice)<750
ORDER BY ProductFinish;
```

26. *Often, when specific (and well-defined) views of the database can be anticipated to be needed on a repeated basis, you can create a VIEW. Also, for simplicity and for security. This also really introduces joins.*

What are the data elements necessary to create an invoice for a customer? Save as a view.

```
CREATE VIEW Invoice_V AS
SELECT Customer_T.CustomerID, CustomerAddress, Order_T.OrderID,
       Product_T.ProductID, ProductStandardPrice, OrderedQuantity
FROM Customer_T, Order_T, OrderLine_T, Product_T
WHERE Customer_T.CustomerID=Order_T.CustomerID
AND Order_T.OrderID = OrderLine_T.OrderID
AND Product_T.ProductID=OrderLine_T.ProductID;
```

Note: Look at the left menu bar under Views. You will see Invoice_v.

27. What are the data elements necessary to create an invoice for order number 1004?

```
SELECT CustomerID, CustomerAddress, ProductID, OrderedQuantity
FROM invoice_v
WHERE OrderID=1004;
```

A view cannot be updatable when:

- 1 The SELECT clause includes the keyword DISTINCT
- 2 The SELECT clause contains expressions, including derived columns, aggregates, functions, etc.
- 3 The FROM clause, a subquery of a UNION clause references more than one table
- 4 The FROM clause or a subquery references another view that is not updateable.
- 5 The CREATE VIEW command contains a GROUP BY or a HAVING clause.

Dynamic view: temporarily created each time it's used. Like a query.

Materialized view: data for that view is actually stored.