SQL-LAB06-View-Join

Purpose of Lab:

1 Joins – done briefly in previous lab

2 Views

3 Copying data from one table to another

# Part 1: Review of Joins

Producing a report that requires data from CUSTOMER and SALESREP from PREMIERE

Fields:

Customer\_Number

First\_Name of the customer

Last\_Name of the customer

First\_Name of the sales representative

Last\_Name of the sales representative

Tables:

PREMIERE.CUSTOMER, PREMIERE.SALESREP

The Salesrep data to show would be the salesrep that services that customer. To do that requires a common field. We will use the common column Sales\_Rep\_Number as it is found in both tables.

The Sales\_Rep\_Number is the primary key in the SALESREP table

The Sales\_Rep\_Number is the foreign key in the Customer table.

When we put a row of the CUSTOMER table together with the corresponding row of the SALESREP table we get a temporary table called a join.

We can JOIN CUSTOMER with SALESREP with a SELECT statement:

SELECT c.CUSTOMER\_NUMBER, c.FIRST\_NAME, c.LAST\_NAME, s.FIRST\_NAME, s.LAST\_NAME

FROM PREMIERE.CUSTOMER c, PREMIERE.SALESREP s

WHERE c.SALES\_REP\_NUMBER = s.SALES\_REP\_NUMBER

ORDER BY c.CUSTOMER\_NUMBER

NOTE:

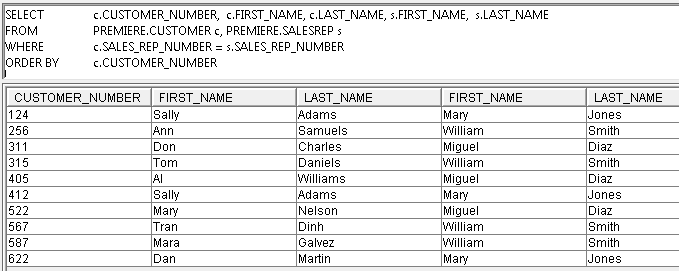
The characters c and s are aliases.

c stands for PREMIERE.CUSTOMER

s stands for PREMIERE.SALESREP

They make it shorter to write the SELECT and other statements

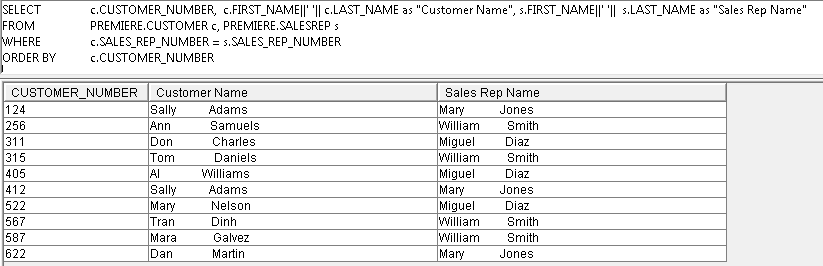
For those using iNavigator the result will be



What is the problem with this report?

We can tell because we wrote the SQL, but for a user the first and last name column headings will tend to cause confusion.

Try concatenating the first and last name of customer and calling it Customer Name and then do the same for sales rep. The result should look like



Once again the concatenation is not as expected because when the name fields were created they were defined as CHAR which means a fixed length field. They should have been defined as VARCHAR to avoid that problem. The next subject DBS301 class will cover how to TRIM off the extra spaces when something is defined as CHAR. As an aside, there are lots of fields that should be char. For example subject code at Seneca is DBS201, IPC144 and has a fixed length of content.

# PART 2: Creating a View

In the previous section, you went to a lot of trouble to generate a report using multiple files, concatenation, etc. What if you or others had to run and execute that statement many, many times? It is possible to save that select statement as a *View*.

The general syntax to create a view is:  
  
CREATE VIEW *viewname*

AS SELECT (insert your select statement here).

Use the same SELECT that you used to create the previous report. Insert before the select the following: CREATE VIEW PREM001.CUST\_REP\_VIEW AS

CREATE VIEW PREM001.CUST\_REP\_VIEW AS

SELECT c.CUSTOMER\_NUMBER, c.FIRST\_NAME||' '|| c.LAST\_NAME as "Customer Name",

s.FIRST\_NAME||' '|| s.LAST\_NAME as "Sales Rep Name"

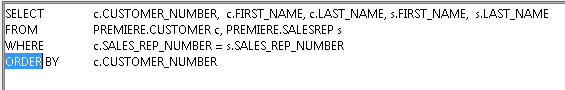
FROM PREMIERE.CUSTOMER c,

PREMIERE.SALESREP s

WHERE c.SALES\_REP\_NUMBER = s.SALES\_REP\_NUMBER

ORDER BY c.CUSTOMER\_NUMBER;

And you received an error as follows:



ORDER BY cannot be used when creating a VIEW. It is at the time of executing the view that the user can determine the order they require.

Correct the statement above and execute.

Once a view is created, it can be used in much the same way as a TABLE. You can use SELECT statements to see the data. Run the select statement to look at all the data in the view.

SELECT \*

FROM PREM001.CUST\_REP\_VIEW;

Your report should look the same as shown in step 1.

**To remove a VIEW**

DROP VIEW PREM001.CUST\_REP\_VIEW;

# Part 3: Copying data from one table to another

You will often need to copy the data in one table to another. In the case of DBAs or Database Administrators, they do it often between databases and servers. For our purposes we will keep it simple.

**Copy 1 table to another**

Do not enter this part; it is for reading and understanding only

Assumption that both tables exist and they have the same structure

INSERT INTO CUSTOMER2 – the receiving table

SELECT \*  
FROM CUSTOMER1; -- the original table of data

Copying, but only some of the rows

INSERT INTO CUSTOMER2

SELECT \*  
FROM CUSTOMER1

WHERE CREDIT\_LIMIT > 10000;

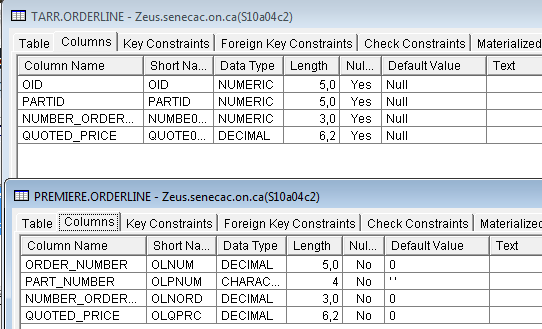
Note: The structures of the tables must match. For example if you created PART\_NUMBER in the new table as numeric but it was CHAR in the original table, it will not work.

This did not work on my example as you can see the structures vary quite substantially

INSERT INTO TARR.ORDERLINE (OID, PARTID, NUMBER\_ORDERED, QUOTED\_PRICE)

SELECT \* FROM PREMIER.ORDERLINE

See the definitions for both as follows:



We will need to change the PARTID column from NUMERIC (5) to CHAR (4) or build them the same at the start.

This is the part you need to do

In order to do this you have to drop your previous customer table.

**DROP TABLE PREM001.CUSTOMER;**

Then issue the following to recreate the CUSTOMER table, but with different data types.

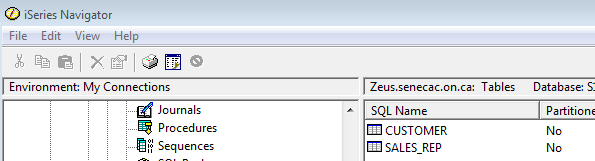
**CREATE TABLE TARR.CUSTOMER(CUSTOMER\_NUMBER CHAR (3) PRIMARY KEY,LAST\_NAME CHAR(15) NOT NULL,FIRST\_NAME CHAR(15) NOT NULL,STREET CHAR(15) NOT NULL,CITY CHAR(15) NOT NULL,STATE CHAR(2) NOT NULL,ZIP\_CODE DECIMAL (5,0) NOT NULL,BALANCE DECIMAL (6,2) DEFAULT 0,CREDIT\_LIMIT DECIMAL (5,0) DEFAULT 0,SALES\_REP\_NUMBER CHAR (2));**

Insert data from premiere as in the example on previous page

Verify it worked with the following:

**SELECT \* FROM PREM001.CUSTOMER;**

In order to see the table on the navigator screen you may have to refresh it.



TO SUBMIT:

Do the following

Write the answer of following question and show it personally next week to get marks during lab period.

**(1)** Execute the following by cutting and pasting the rest of this SQL script. BUT … the first time through remove the DROP table. If you need to do it the full script again you will need the drop tables.

**DROP TABLE PREM001.SALESREP;**

**CREATE TABLE PREM001.SALESREP ( SALES\_REP\_NUMBER CHAR(2) NOT NULL DEFAULT '' , LAST\_NAME CHAR(15) NOT NULL DEFAULT '' , FIRST\_NAME CHAR(15) NOT NULL DEFAULT '' , STREET CHAR(15) NOT NULL DEFAULT '' , CITY CHAR(15) NOT NULL DEFAULT '' , STATE CHAR(2) NOT NULL DEFAULT '' , ZIP\_CODE DECIMAL(5, 0) NOT NULL DEFAULT 0 , COMMISSION DECIMAL(7, 2) NOT NULL DEFAULT 0 , RATE DECIMAL(3, 2) NOT NULL DEFAULT 0 , CONSTRAINT TARR.SALESREPPK PRIMARY KEY( SALES\_REP\_NUMBER )**

**) ;**

**DROP TABLE PREM001.ORDERLINE;**

**CREATE TABLE PREM001.ORDERLINE ( ORDER\_NUMBER DECIMAL(5, 0) NOT NULL DEFAULT 0 , PART\_NUMBER CHAR(4) NOT NULL DEFAULT '' , NUMBER\_ORDERED DECIMAL(3, 0) NOT NULL DEFAULT 0 , QUOTED\_PRICE DECIMAL(6, 2) NOT NULL DEFAULT 0 , CONSTRAINT TARR.ORDERL\_PK**

**PRIMARY KEY( ORDER\_NUMBER , PART\_NUMBER )**

**) ;**

**DROP TABLE PREM001.PART;**

**CREATE TABLE PREM001.PART ( PART\_NUMBER CHAR(4) NOT NULL DEFAULT '' , PART\_DESCRIPTION CHAR(15) NOT NULL DEFAULT '' , ON\_HAND DECIMAL(3, 0) NOT NULL DEFAULT 0 , CLASS CHAR(2) NOT NULL DEFAULT '' , WAREHOUSE DECIMAL(1, 0) NOT NULL DEFAULT 0 , PRICE DECIMAL(6, 2) NOT NULL DEFAULT 0 , CONSTRAINT TARR.PART\_PK PRIMARY KEY( PART\_NUMBER ) ) ;**

**DROP TABLE PREM001.ORDERS;**

**CREATE TABLE PREM001.ORDERS ( ORDER\_NUMBER DECIMAL(5, 0) NOT NULL DEFAULT 0 , ORDER\_DATE DATE NOT NULL DEFAULT CURRENT\_DATE , CUSTOMER\_NUMBER CHAR(3) NOT NULL DEFAULT '' , CONSTRAINT TARR.ORDER\_PK PRIMARY KEY( ORDER\_NUMBER ) ) ;**

**-- copying data from premiere**

**-- INSERT INTO PREM001.CUSTOMER -- already done above**

**-- SELECT \* FROM PREMIERE.CUSTOMER;**

**INSERT INTO PREM001.ORDERS**

**SELECT \* FROM PREMIERE.ORDERS;**

**INSERT INTO PREM001.PART**

**SELECT \* FROM PREMIERE.PART;**

**INSERT INTO PREM001.ORDERLINE**

**SELECT \* FROM PREMIERE.ORDERLINE;**

**INSERT INTO PREM001.SALESREP**

**SELECT \* FROM PREMIERE.SALESREP;**

**ALTER TABLE PREM001.ORDERLINE ADD CONSTRAINT PREM001.ORDERLFK1 FOREIGN KEY( ORDER\_NUMBER ) REFERENCES PREM001.ORDERS ( ORDER\_NUMBER ) ;**

**ALTER TABLE PREM001.ORDERLINE ADD CONSTRAINT PREM001.ORDERLFK2 FOREIGN KEY( PART\_NUMBER ) REFERENCES PREM001.PART ( PART\_NUMBER ) ;**

**ALTER TABLE PREM001.ORDERS ADD CONSTRAINT PREM001.ORDERSFK1 FOREIGN KEY( CUSTOMER\_NUMBER ) REFERENCES PREM001.CUSTOMER ( CUSTOMER\_NUMBER ) ;**

# (2) VIEWS – going back to VIEWS with your own data

Recreate the view before except using the tables you have in your database and not the ones in PREMIERE.

Then verify it works with the following:

**SELECT \***

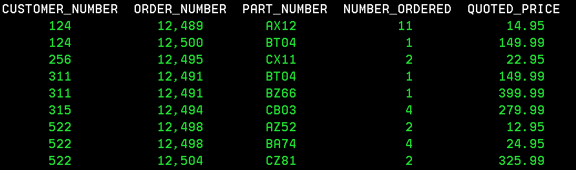
**FROM PREM001.CUST\_REP\_VIEW;**

**Create a view in your collection called ORDER\_INFO**

The view shows the order information for each customer in the collection. The report should contain the Customer Number and the Order Number from the Order table and the Part Number, Number Ordered and Quoted Price from the Orderline table.

**REMEMBER: when you select columns from more than 1 table, you must tell it which column to match on using the "WHERE" clause.**

The report should look like this:

****

It would be nice if we could see a total value for each order line. You have learned in the normalization process that we do NOT store calculated values. Instead of taking up valuable storage space, we can create computed fields using SQL whenever we need them. Unfortunately, we cannot alter a view once it has been created. We could drop the view and recreate it with the new requirements or create a new view.

Create a new view called FULL\_ORDER\_INFO

The report will look like this or the equivalent

