**Purpose:**

* **To learn about constraints**
* **To learn about a Foreign Key Constraint**
* **How to establish 1:M relationships between tables using Foreign Keys**
* **The syntax of the command to identify the Foreign Key**
* **How the database preserves referential integrity using Foreign Keys**
* **To learn about a Unique Key Constraint**
* **To learn about a Check Key Constraint**
* **To learn how to test if our constraints are working**
* **To be able to print out information on Table/File Constraints**

### Requirements to pass the lab:

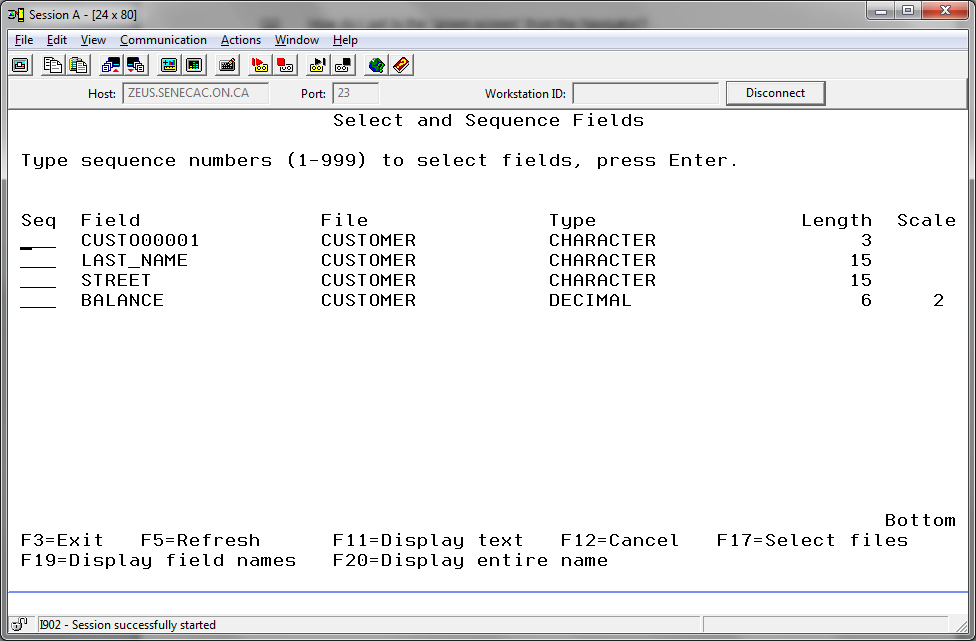
* **Hand in a system generated printout of your CUSTOMER table constraints correctly applied with your name printed at bottom page**
* **Hand in a system generated printout of your SALESREP table constraints correctly applied with your name printed at bottom page and the appropriate data in the table**
* **Answer the questions on the answer sheet**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In lab 2 you created two tables in a collection called PREMIER??? Where ??? are the last three characters of your user profile.

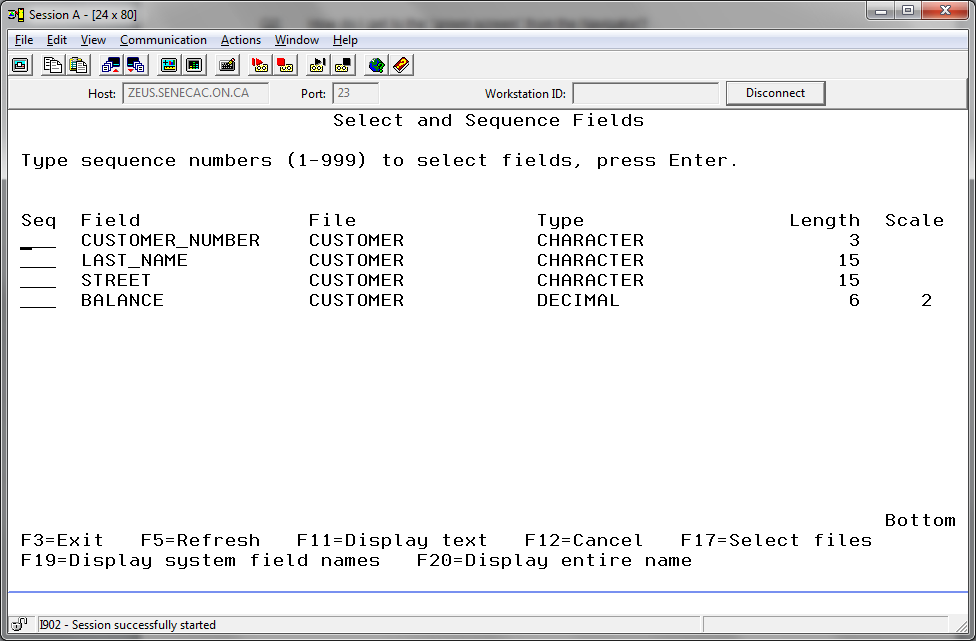
Today we want to work with the tables in this collection, but we do not want to continually have to type the collection name as a qualifier. On the answer sheet indicate what statement will relieve us of having to type PREMIER???.CUSTOMER and PREMIER???.SALESREP and instead just refer to CUSTOMER and SALESREP during our interactive SQL session?

If you have forgotten the field names you used for the customer table, we know that we can type SELECT (F4) enter the table name and then press F4 on the fields prompt. Do this



You did not name the first field or column CUSTO00001. That is a tricky name to type because it has the letter O mixed in with a few zeroes(0) and you could enter that incorrectly. You can avoid this problem by putting a ‘1’ beside the field name to select, but you want to use the column name you provided on your CREATE TABLE statement.

If you prsss F19 (Shift + F7) you will see the more friendly column name.



To Delete rows from a table using IBMi interactive SQL

In the previous lab, we created 2 tables: CUSTOMER and SALESREP and entered a couple of rows of data into each one. We will now delete that data using Client Access for customer and using the Navigator for SALESREP.

1. Start a session of Access Client Solutions

The general format of the command to delete a row is:

DELETE FROM table-name

WHERE (condition)

For example; DELETE FROM EMPLOYEE

WHERE EMPLOYEE\_NUMBER = ‘187’

Using the DELETE command delete the row of data in CUSTOMER that doesn’t have your name.

You may need to reacquaint yourself with the customer numbers in each row.

Indicate on the answer sheet the statement that would achieve this purpose?

You can remove all the rows in the customer table by not specifying a WHERE clause with your DELETE statement. You should be very careful when running a DELETE statement without a WHERE clause when you are on the job.

It is okay in this learning environment, so use that statement for the remaining row in the CUSTOMER table.

(assuming you entered at least two rows in lab 1)

Now we will delete all the rows of data in SALESREP using a single statement. What is that statement you ran to do this?

To Establish 1:M relationships between tables using foreign keys:

- The relationship between SALESREP and CUSTOMER is 1:M.

- This means that one particular salesrep has many customers but,

- one particular customer has one and only one salesrep.

- When we have found the information about a customer, we sometimes want information about that customer’s salesrep. To get the salesrep information we will use a field on the customer row called Sales\_Rep\_Number. We use that value to obtain information about the customer’s salesrep from the SALESREP table.

- Did you notice that the SALESREP table does not have a Customer\_Number field ?

- That wouldn’t work! How would we represent the fact that one salesrep has many customers.

We could only enter a single customer number in that field - not many customers.

🡺 Definition: A Foreign Key is a field on one table that matches the Primary Key of another table.

- The Foreign Key field is always placed on the "many" table when there is a 1-to-many relationship between 2 tables..

On the answer sheet fill in the blanks to summarize what you have just learned.

Here is an example of the Syntax of the command that sets up the Foreign Key

In our example, a Family Doctor has many Patients but that a particular patient has one and only one Family Doctor. The PATIENT table will have the Foreign Key.

Here is the example:

ALTER TABLE PATIENT

ADD CONSTRAINT Doctor\_FK

FOREIGN KEY (Doctor\_Number)

REFERENCES DOCTOR(Doctor\_Number)

Using the above example, execute the same command to create the Foreign Key that will relate the CUSTOMER and SALESREP tables. Read the message above the command line to verify that the constraint has been created.

Indicate on the answer sheet what the statement to do this is?

If your new constraint did not work, here are some of the reasons an attempt to introduce a foreign key constraint may fail

A) CUSTOMER has a Sales\_Rep\_Number that is character 3 and SALESREP has a Sales\_Rep\_Number that is character 2.

B) One table may have a numeric field version that is 2 and the other table has a character field version that is 2.

Note. It is okay for different names to be used for this field or column in both tables as long as the size and type agree.

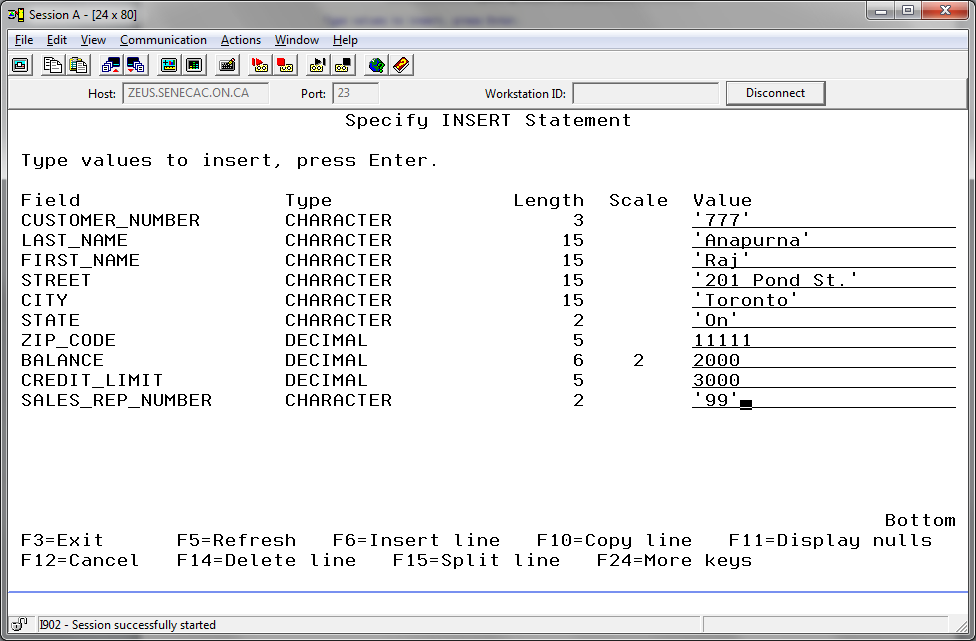
How the DBMS preserves referential integrity using Foreign Keys.

🡺 Definition: Referential Integrity: When we want to add a new Customer, we must specify a Salesrep\_Number of a Salesrep that exists on the SALESREP table.

We cannot “refer” to a Salesrep that does not exist!

Let’s try to add a new Customer who has a Salesrep that doesn’t exist. The database should refuse to add the new row. Let’s add a row to the CUSTOMER table with the data as shown below.

Use ‘Anapurna’ for use LAST\_NAME and ‘Raj’ for FIRST\_NAME and ‘99’ for SALES\_REP\_NUMBER.



Type:

**INSERT INTO CUSTOMER**

**VALUES (‘777’, ‘Anapurna’, ‘Raj’, ‘201 Pond Street’, ‘Toronto’, ‘ON’,**

**11111, 2000, 3000, ‘99’)**

Press Enter to submit. An error message will be displayed.

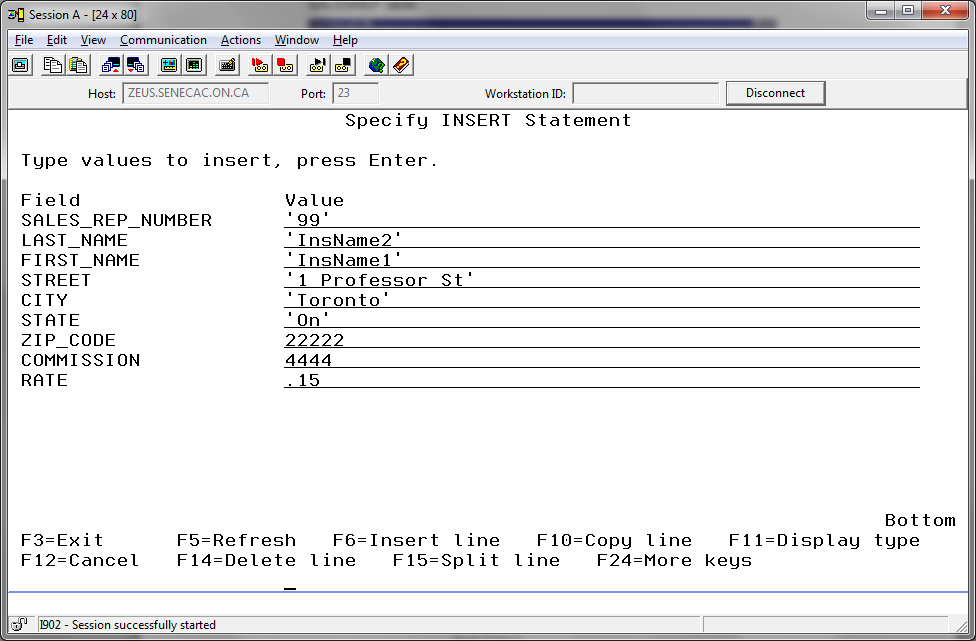
Put your cursor on the error message, press the help key (F1) and read the explanation.

Indicate on the answer sheet what you think caused this error?

So we must enter Salesrep number ‘99’ to the SALESREP table before we can add Customer ‘777’. Because of the Foreign Key constraint, the database will not allow us to add a Customer who references a Salesrep who does not exist on the SALESREP table.

If we add Salesrep ‘99’ to the SALESREP table, the INSERT into CUSTOMER command that failed before should work. Let’s try it!

Add a row to the SALESREP table with Sales\_Rep\_Number of '99’ and use your instructor’s name and the information shown in the screenshot above.



Now we’ll try the INSERT INTO CUSTOMER that failed before.

Retrieve the statement you had used to try to insert a row in the CUSTOMER table (use F9). Press Enter to run the statement. You should not get an error this time and a row should have been added to your CUSTOMER table.

As you can see the Foreign Key Constraint performs 2 functions:

i. The Foreign Key **links** the two tables together

ii. The Foreign Key preserves **referential integrity** between the 2 tables.

Adding a CHECK constraint.

The five types of constraints are :

i) PRIMARY KEY CONSTRAINT

ii) FOREIGN KEY CONSTRAINT

iii) NOT NULL or NOT NULL WITH DEFAULT

iv) CHECK CONSTRAINT

v) UNIQUE CONSTRAINT

A check constraint establishes a rule about values that are allowed in a particular field. For example, we could establish a check constraint that allows only the values ‘M’ or ‘F’ for a gender column.   
  
Let's add a check constraint to the SALESREP table. We could have done this when we created the table but because the table already exists we will use the ALTER TABLE command.

Example 1:Here is the syntax for a typical CHECK constraint – a range check:

**ALTER TABLE EMPLOYEE**

**ADD CONSTRAINT Salary\_CK**

**CHECK (Salary Between 20000 and 40000)**

If the user attempts to change his friend’s salary to 50000, the database will refuse to make the change because it is not in the

range of acceptable values. (Note: 20000 and 40000 would be accepted.)

Example 2: Another type of check constraint ensures that the field value is in a list.

**ALTER TABLE PREMIER???.STUDENT**

**ADD CONSTRAINT PREMIER???.Grade\_CK**

**CHECK (Grade IN (‘A’, ‘B’, ‘C’, ‘D’, ‘F’))**

In this case if the user enters a Grade of **‘E’ ,** the database will refuse to make the change because it is not in the list of acceptable values.

Notice in the first example the default schema is used. In the second example both the table **and the constraint name** are qualified with the schema name.

Let’s try constructing a constraint based on a LIST :

Field Name: Rate

Table Name: SALESREP

Constraint Type: CHECK

Constraint Name: Rate\_CK

Constraint Condition: RATE must equal 0.05 or 0.06 or 0.07 or 0.15

Remember: The IN keyword is used for a list of values. Run the statement to do this and write the constraint on the answer sheet.

Make sure that you receive the completion message "Alter completed for table SALESREP in … ".

Let’s try a constraint based on a RANGE. Write the constraint on the answer sheet.

Field Name: Balance

Table Name: CUSTOMER

Constraint Type: CHECK

Constraint Name: Balance\_CK

Constraint Condition: Balance must be greater than 0 and less than 6001

Hint: We could use (BALANCE > 0 AND BALANCE < 6001)

We could also use the BETWEEN 1 AND 6000

Adding a UNIQUE constraint

A UNIQUE constraint on a column permits a value to occur only once in that column in the entire table.

Example:

Assume that a city can have only one sales rep assigned. The values in the city column in the SALESREP table must be therefore be unique. We must prevent the user from assigning more than one Salesrep to any particular city.

Field Name: City

Table Name: SALESREP

Constraint Type: UNIQUE

Constraint Name: City\_UN

Constraint Condition: A city can be assigned to one and only one Salesrep

**ALTER TABLE PREMIER???.SALESREP**

**ADD CONSTRAINT PREMIER???.City\_UN**

**UNIQUE (City)**

Once again, make sure you receive the "Alter completed" message.

Let’s write the UNIQUE constraint for an **imaginary** EMPLOYEE table on the answer sheet:  
  
Field Name: Ontario\_Medicare\_Num

Table Name: EMPLOYEE

Constraint Type: UNIQUE

Constraint Name: Ontario\_Medicare\_Num\_UN

Constraint Condition: Any particular Medicare Number can occur only once on the table.

***Note: SQL does not allow a constraint to be modified – you have to drop and re-enter the constraint.***

* + 1. How to test if our constraints are working.

The best way to test things is to set up a test plan. In a test plan we decide:

1. what we want to test
2. what action will test for that feature
3. what we predict the database should do
4. verify the test by performing the action

***We should verify cases where the constraint should allow the new row to be added and cases where the new row is rejected.***

TABLE 1: Testing of Constraints

|  |  |  |  |
| --- | --- | --- | --- |
| **Constraint** | **Action** | **Prediction** | **Did the constraint work?** |
| **Rate\_CK** | Insert a row of data into the SALESREP table with a rate of 0.22 | row should be rejected because the value 0.22 is not in the list of acceptable values | Was the row rejected?  **yes / no** |
| Insert a row of data into the SALESREP table with a rate of 0.05 | row should be accepted as 0.05 is in the list of acceptable values | Was the row accepted?  **yes / no** |
| **City\_UN** | insert a 2 rows of data into the SALESREP table each has value for City: ‘Montreal’ | should not accept the second row because city is UNIQUE and therefore ‘Montreal’ cannot be used twice | Was the row rejected?  **yes / no** |
| insert a row of data into the SALESREP table with a city of ‘Oakville’ | row should be accepted because ‘Oakville’ has not been used before | Was the row accepted?  **yes / no** |
| **Balance\_CK** | insert a row of data into the CUSTOMER table with a Balance = 1555 | row should be accepted – it is in the acceptable range. | Was the row accepted?  **yes / no** |
| insert a row of data into the CUSTOMER table with a Balance = 8000 | row should be rejected because the Balance is more than 8000 which is the maximum allowed | Was the row rejected?    **yes / no** |

If you wanted a printout using Access Client Solutions:

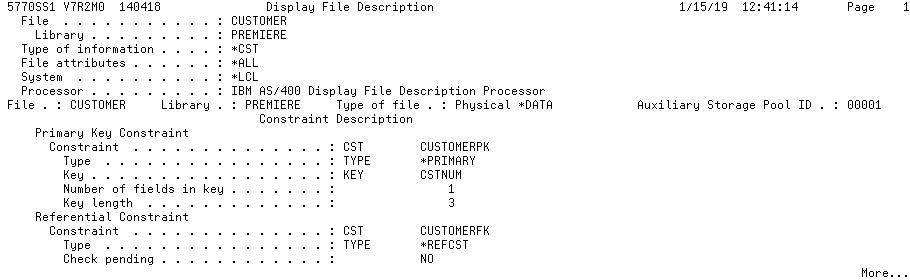
The constraint information is easily available in the native language. **If you exit interactive SQL** you can see all the constraints for CUSTOMER on your display screen with a command similar to the one shown below for the instructor’s sample table:

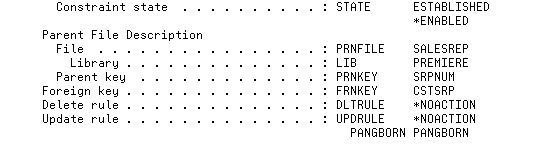
**DSPFD FILE(PREMIERE/CUSTOMER) TYPE(\*CST)**

If you want a printed report for CUSTOMER rather than showing output on the screen, you just need to override where the output is being sent

**DSPFD FILE(PREMIERE/CUSTOMER) TYPE(\*CST) OUTPUT(\*PRINT)**

**Here is the sample report:**





Note all the important information provided by the printout. A proof that constraints have been applied, the names of the table, collection and constraint columns and the types of constraints.

After you have run the reports for your CUSTOMER and SALESREP tables, you can send them to the printer and submit them as proof that your constraints were produced correctly.

You could also run a query in SQL that shows the following for all your constraints in a schema: The object that has this information is one of the ones listed in Lab 1.

SYSCST \*FILE LF SQL catalog view

If you Prompt the SELECT statement with the proper object name, you can find these columns to list.

