SQL Lab – Intro to SQL

This lab is an introduction to working with SQL. In this lab, you will review materials from in-class, and complete hands-on tasks working with SQL. Review each portion of this lab carefully as we will continue to build on this.

# Part 1 – SQL Notation

We discussed in class the standard notation when writing SQL statements. As a review, you should follow these guidelines:

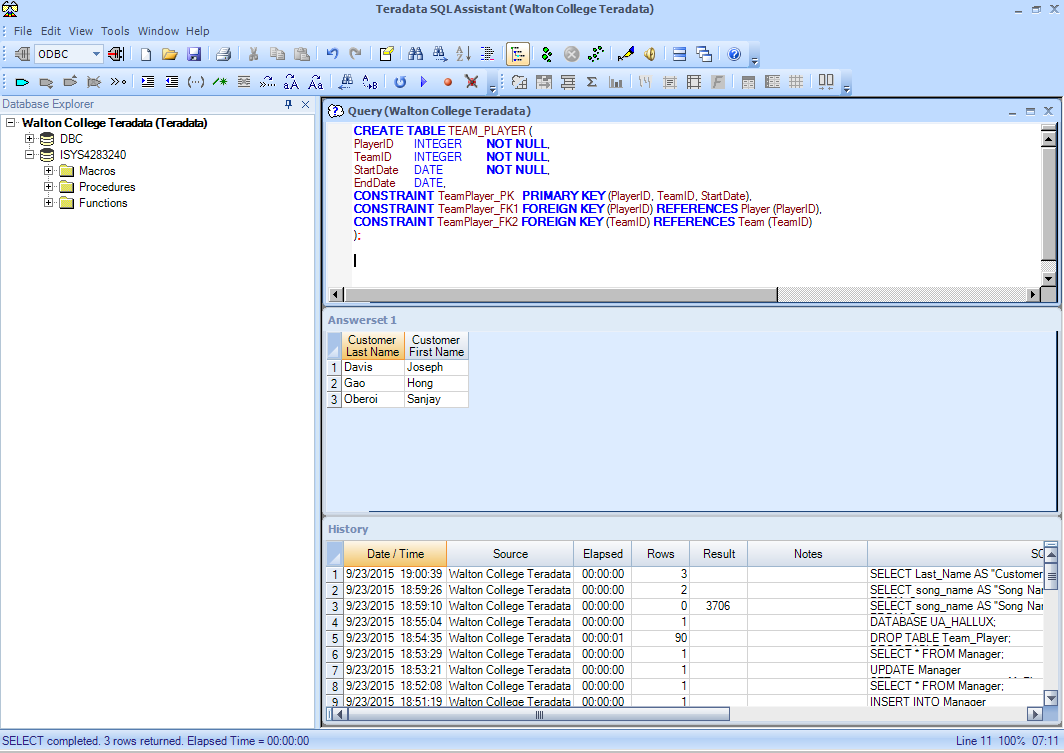
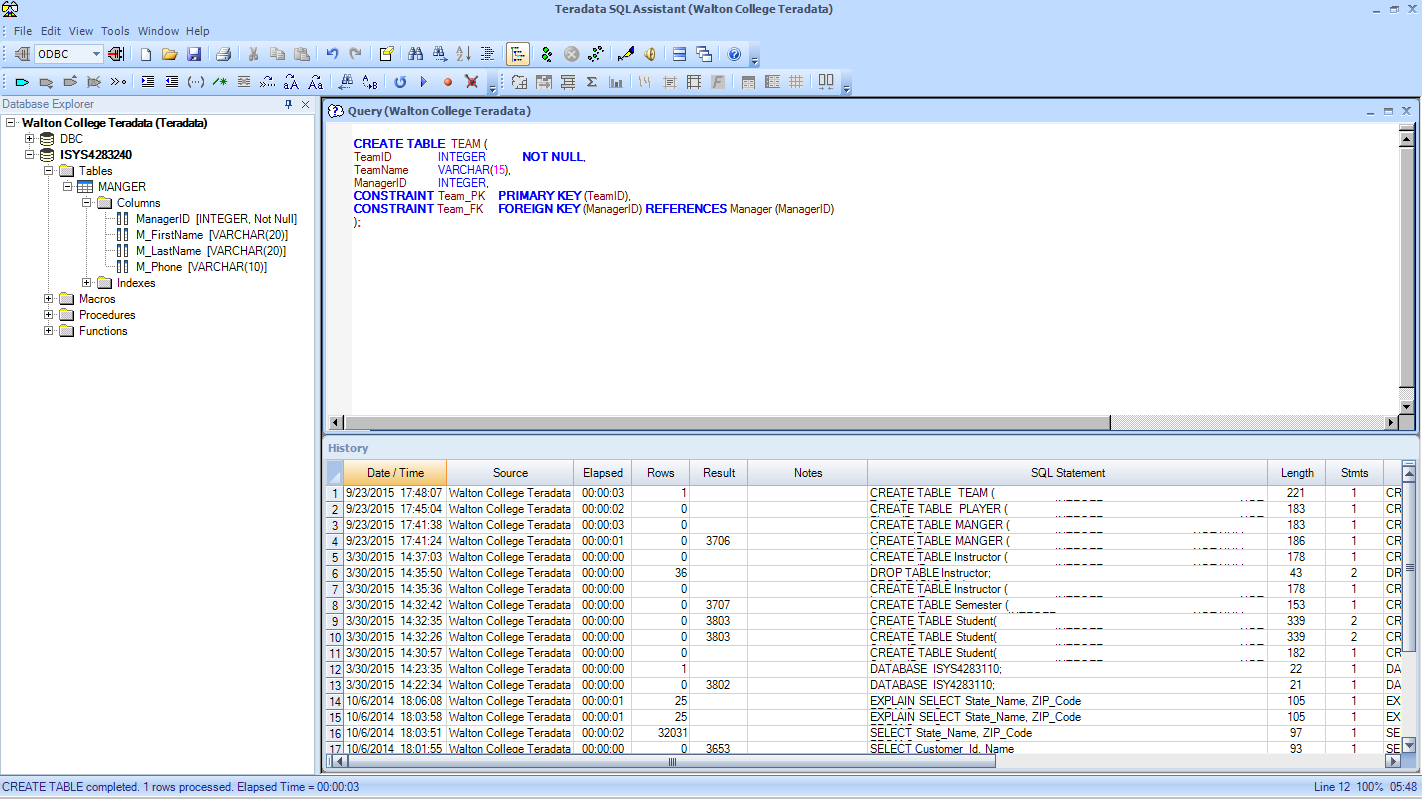
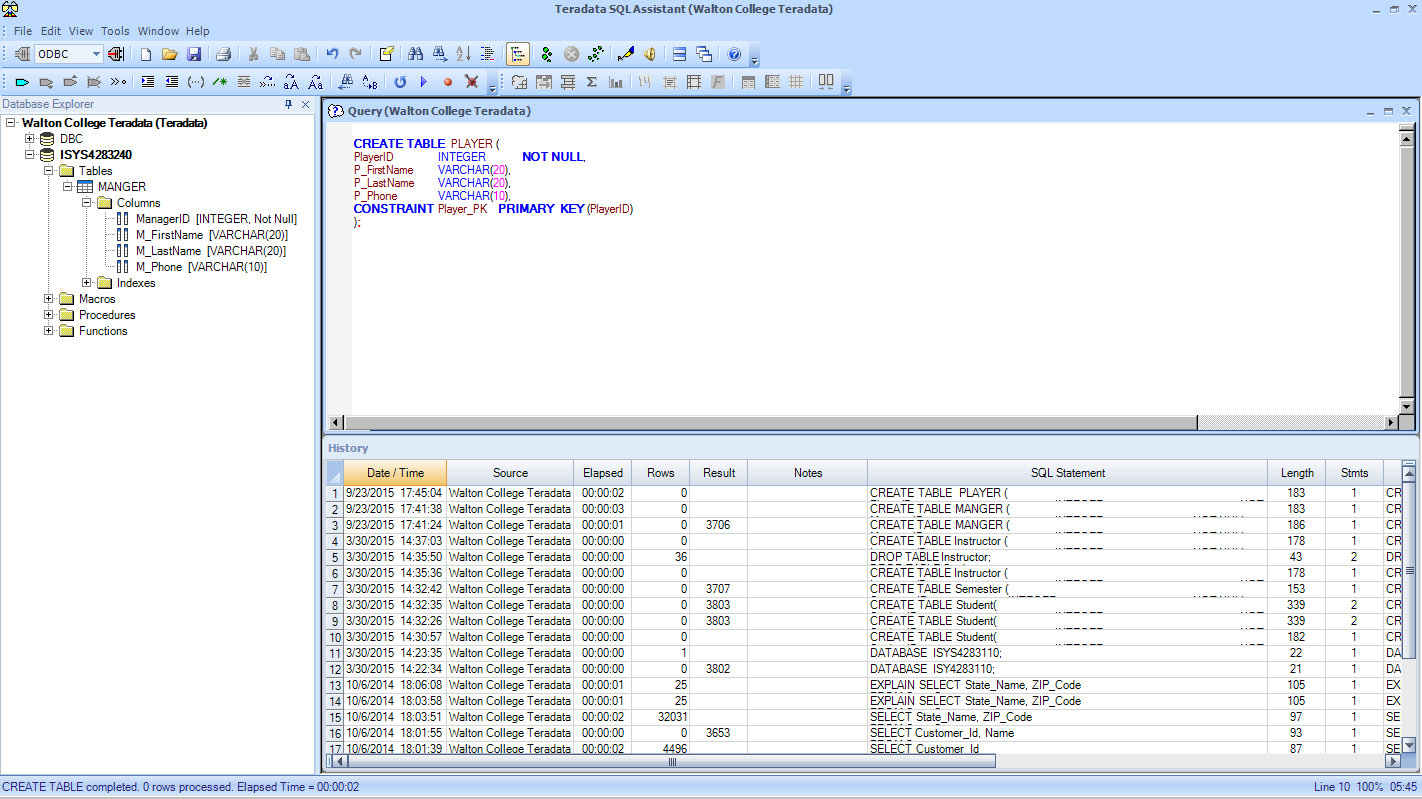
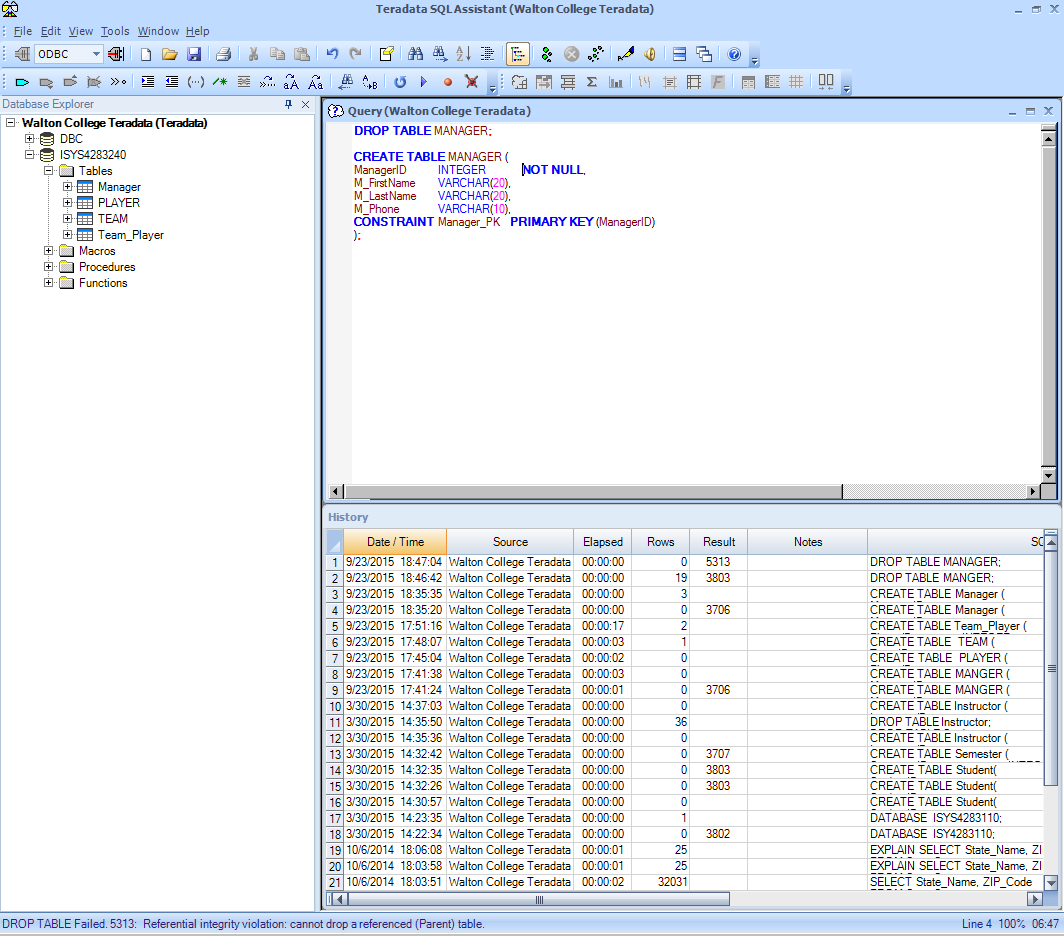
* All SQL keywords (CREATE, SELECT, etc) should be in all caps
  + This is not required by the DBMS; however, it makes reading your SQL much easier
* All table/field names are entered as one word – no spaces
  + Can use mixed-case words (i.e. CustomerName)
* All SQL statements should end in a semi-colon (;)

# Part 2 – Working with Data Types in SQL

We discussed in class the standard data types used for working with data in SQL. The data types discussed included:

|  |  |
| --- | --- |
| Strings | CHAR  VARCHAR or VARCHAR2 |
| Numbers | NUMBER  INTEGER/INT  DOUBLE  FLOT |
| Dates | TIMESTAMP  DATETIME  DATE |

# Part 3 – CREATE Statements in SQL

In class we looked at the example of creating the tables for the baseball team. Your CREATE statements should begin with naming the table, listing each of the fields, their associated data type, any constraints for the individual field, and then adding the PRIMARY KEY and FOREIGN KEY constraints. Below are the tables we created in class.

We also discussed creating an IDENTITY COLUMN for the Manager table where it would auto-increment the primary key field for us for every record that was inserted into the table. The code we used to do this was:

## 

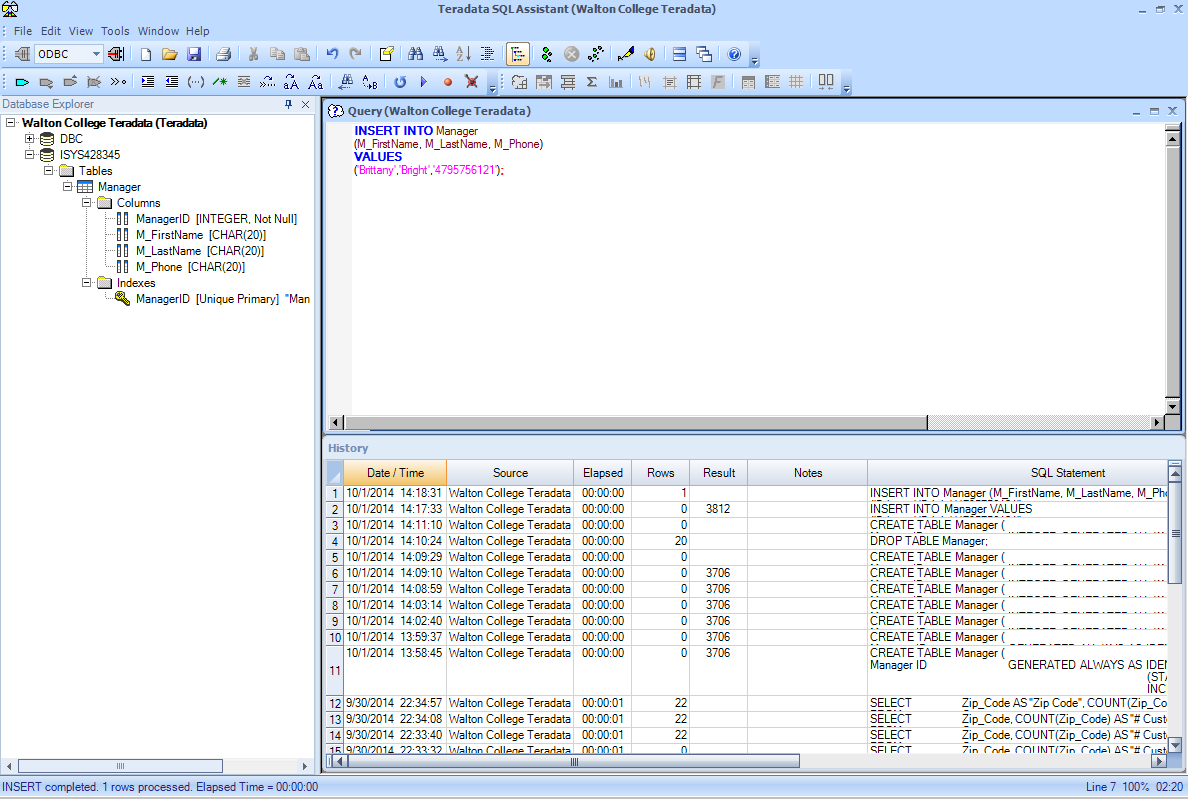
## **Constraints (other than PRIMARY KEY & FOREIGN KEY)**

We talked about adding individual constraints to the fields within a table during class. You should be aware of some of the common types of constraints you could add to a field within your table:

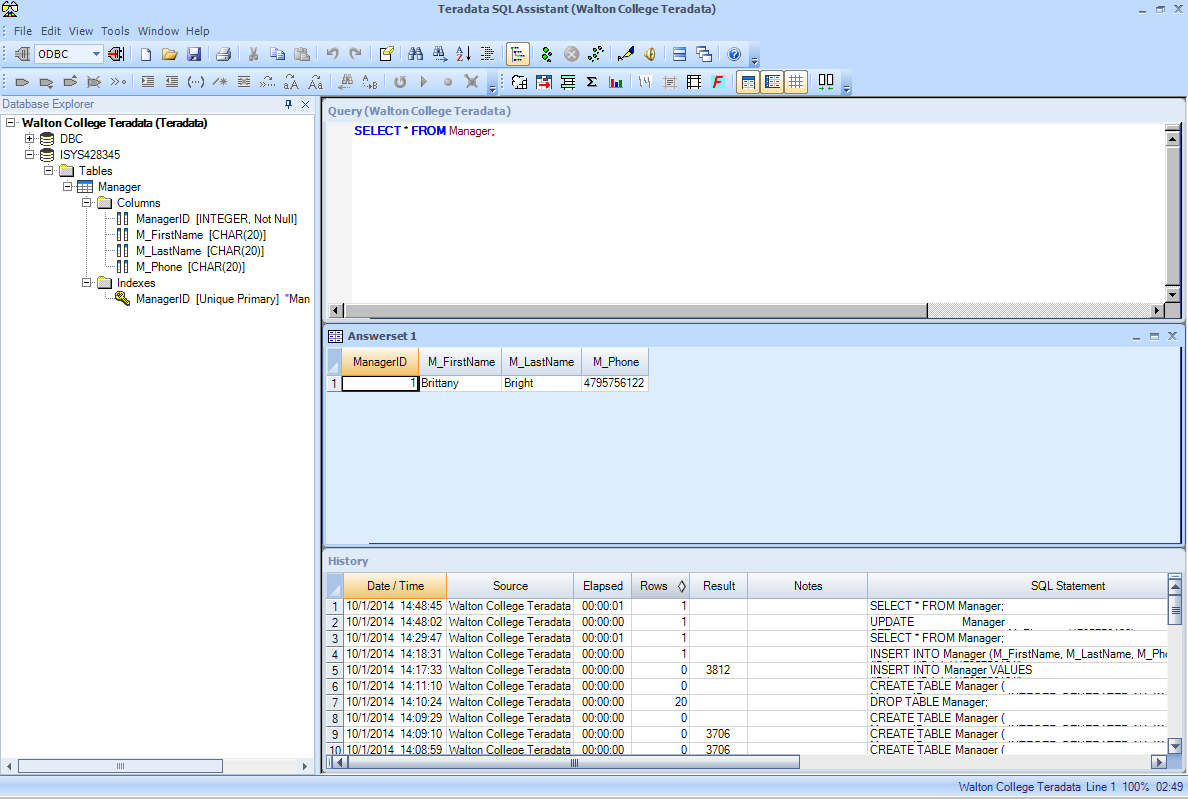
|  |  |  |
| --- | --- | --- |
| Constraint | Purpose | Example |
| NOT NULL | Enforces that the field must contain a value. | NOT NULL |
| CHECK | Enforces that the field must contain a value in this list. | CHECK(ProductFinish IN(‘Cherry’,’Natural Ash’, ‘White Ash’)) |
| DEFAULT | If the field is left null, this value will be stored in the field. | DEFAULT SYSDATE |

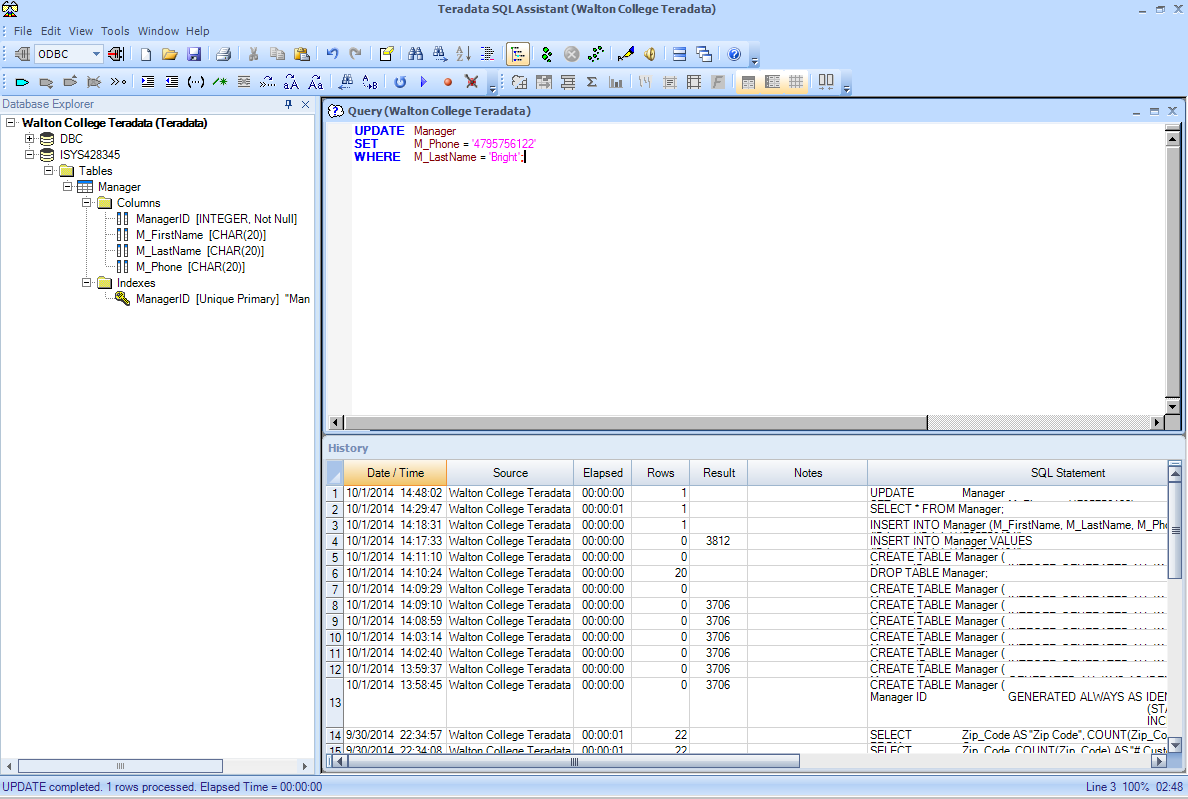
# Part 4 – INSERT & UPDATE Statements in SQL

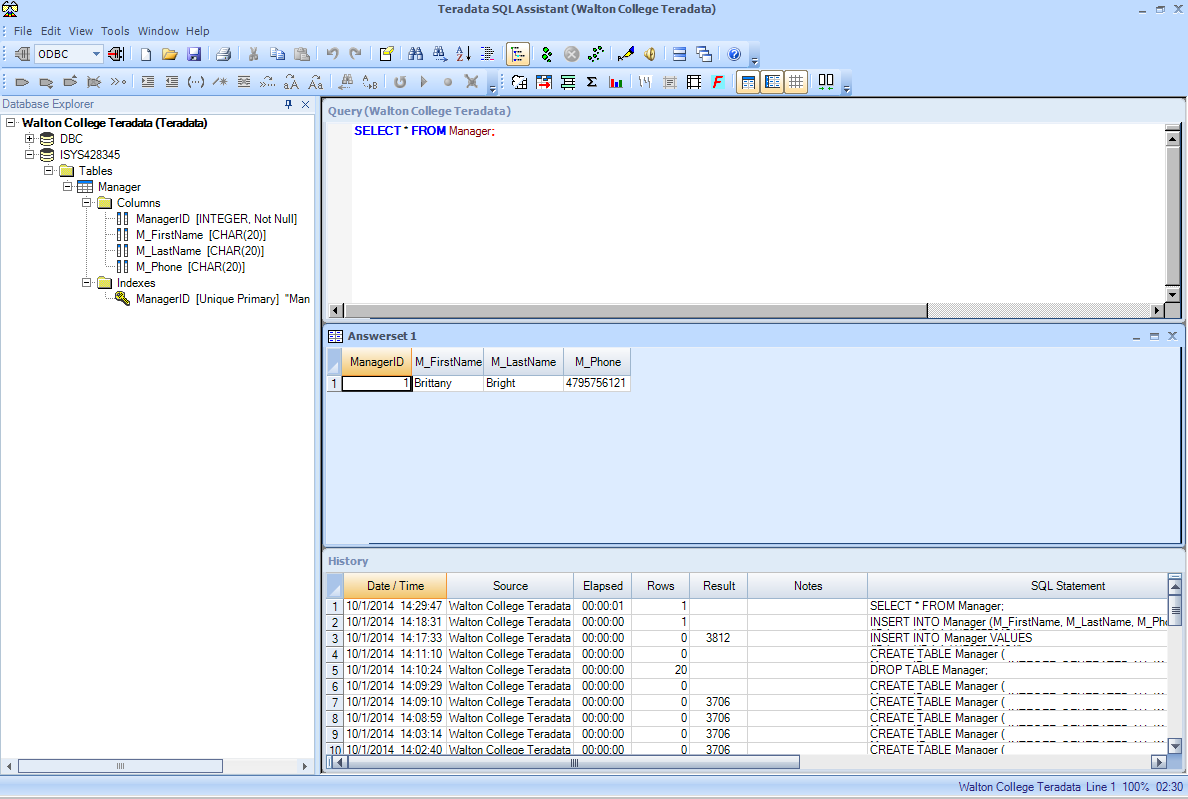
Next, it is important to understand inserting records into your tables. This is one of the SQL statements that you will most likely use most frequently within your programming. You must add your daily transaction information to your database, right?

We discussed in our lecture two methods of inserting data into a table. Here, we will only discuss the one, preferred, method of inserting into data into a table. Below you will find the example syntax for inserting a record into your Manager table (from above).

Why did I not include my ManagerID in my list of columns? Since we have added an IDENTITY COLUMN to the Manager table for the ManagerID field, it will auto increment this value for me. Therefore, the contents of my table will be:



Oh no, that wasn’t the correct phone number in my insert statement! How do I fix that? You would use the UPDATE statement within SQL.

The contents of my record within my table would now be:

# Part 5 – Specifying a DATABASE

When working with SELECT statements in SQL, you must specify to the DBMS which database you’re going to be using within your statements. You either must do this within every statement you write, or you can do it up front and tell the DBMS I will be using it from this point forward until I tell you otherwise. This is something that will vary from DBMS to DBMS as far as the syntax used. For SQL Server, you will use:

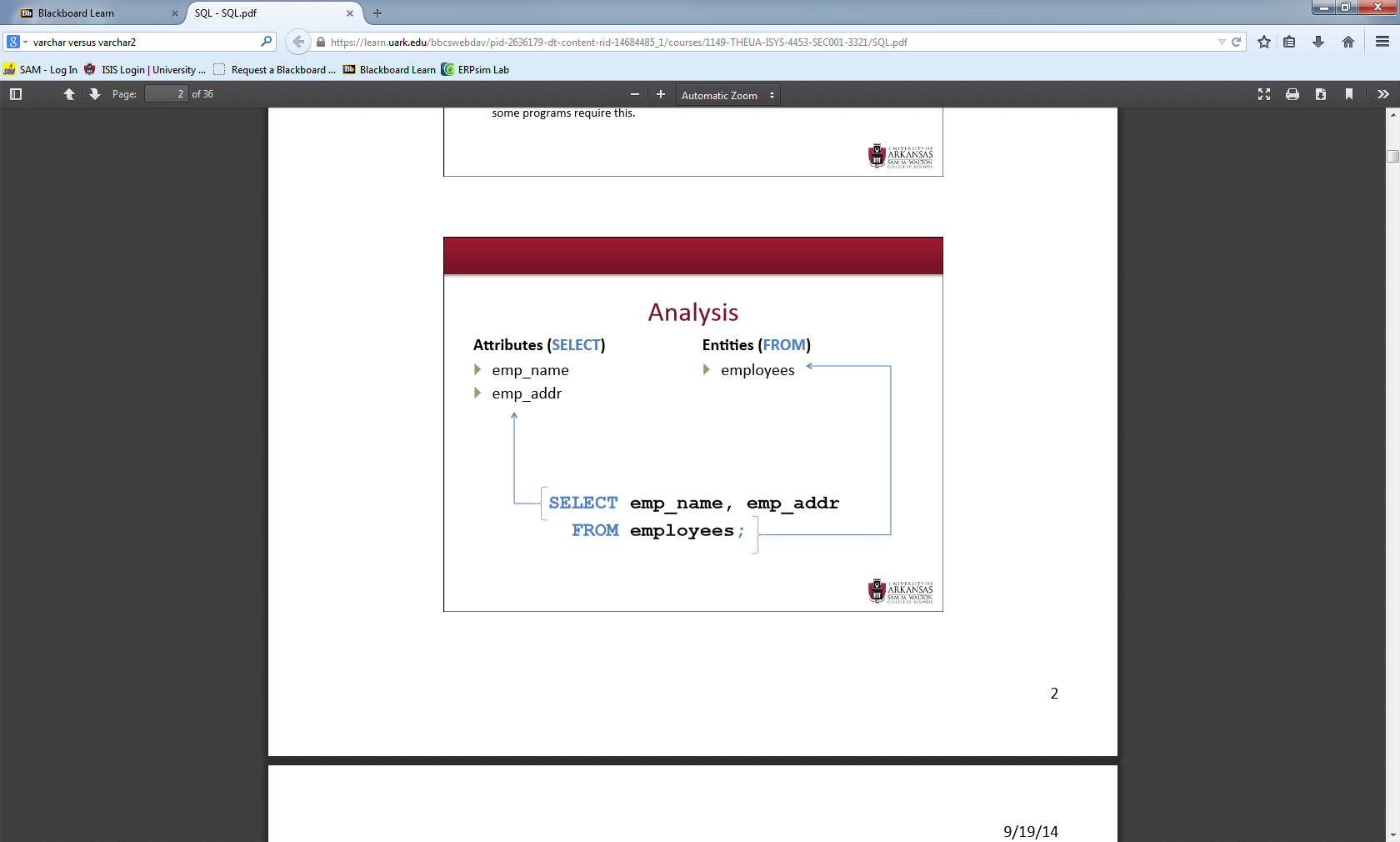
**USE** DatabaseName;

For example, in the exercises below you will be working with a database called UA\_HALLUX. To tell the DBMS that you will be working with this database, you would execute the command

**USE** UA\_HALLUX;

Essentially, this sets the current default database to UA\_HALLUX.

# Part 6 – SELECT Statements in SQL

Our SELECT statement has two components, our attributes and our entities.

In addition, we talked about two optional components to any select statement, the WHERE and the ORDER BY. The WHERE clause allows for the conditions of our statement. The ORDER BY allows sorting of the records returned.

SELECT emp\_name,  
 emp\_addr  
FROM employees  
**WHERE** emp\_name = ’Bright’  
**ORDER BY** emp\_name;

If you recall, we discussed in your WHERE clause you could use the basic mathematical operators (greater than, less than, equal to, not equal to), the BETWEEN \_\_\_\_ AND \_\_\_\_\_, your Boolean operators (AND, OR, NOT), the use of LIKE and the wildcard (%), the IN operator, and IS NULL. Be sure to review these in the PowerPoint slides from chapter 6.

## **Aliases**

In class we also discussed the use of both a table and a column alias to shorten the name used to refer back to them within your SQL. You will notice in the example below that your alias follows AS in the column area and it follows your table name in the table area. Your table alias becomes useful when we begin looking at joins in the next chapter, working with multiple tables in the same SELECT statements.

SELECT C.CustomerName **AS “Name”,**  
 C.CustomerAddress **AS “Address”**  
FROM Customer\_V **C**WHERE Name = ‘Home Furnishings’;

# Part 7 – Aggregate Functions in SQL

If you recall, your aggregate functions are when you use the functions within your SELECT statement to summarize your data (i.e. COUNT). Below you will find a list of the most common aggregate functions used in SQL statements.

|  |  |
| --- | --- |
| Aggregate Function | Purpose |
| AVG(columnName) | Provides you with the average of the column you provide. |
| COUNT(columnName) | Provides you with the count of the column or rows you provide. |
| MAX(columnName) | Provides you with the maximum value of the column you provide. |
| MIN(columnName) | Provides you with the minimum value of the column you provide. |
| SUM(columnName) | Provides you with a sum value of the column you provide. |

When using aggregate functions, we discussed you could have two types of results (scalar and vector). Your scalar results would only return one result (i.e. COUNT); however, your vector results would return multiple rows and would require the use of the GROUP BY command. We also discussed the ability to apply conditions to your groups using the HAVING clause. Be sure to review these examples in the Chapter 6 slides from lecture.

# Part 8 – Converting Data Types in SQL

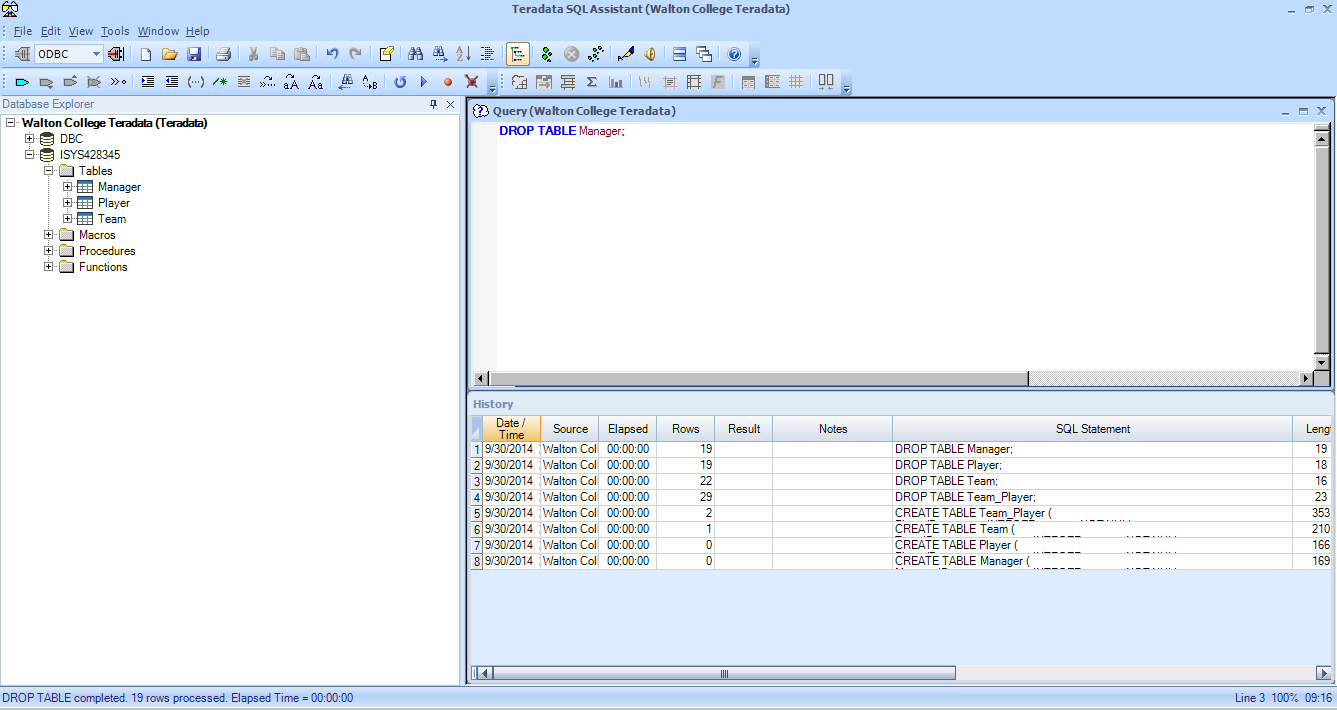
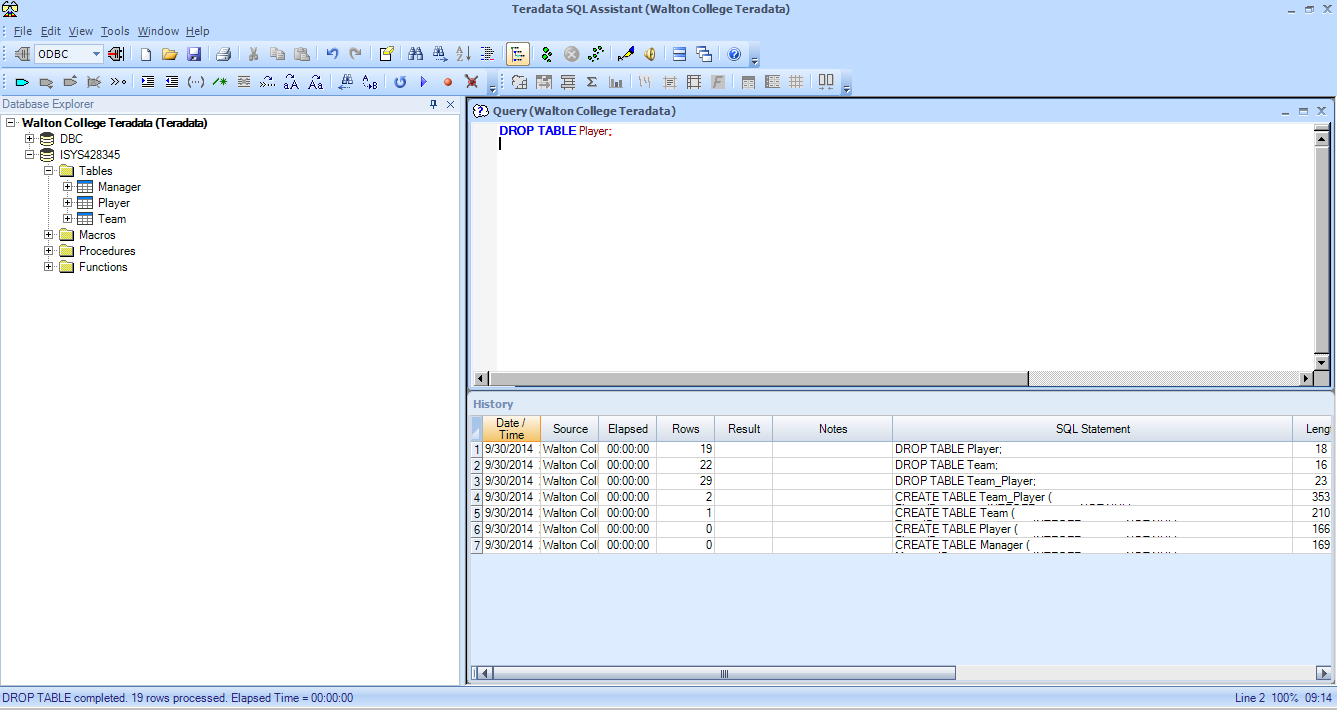
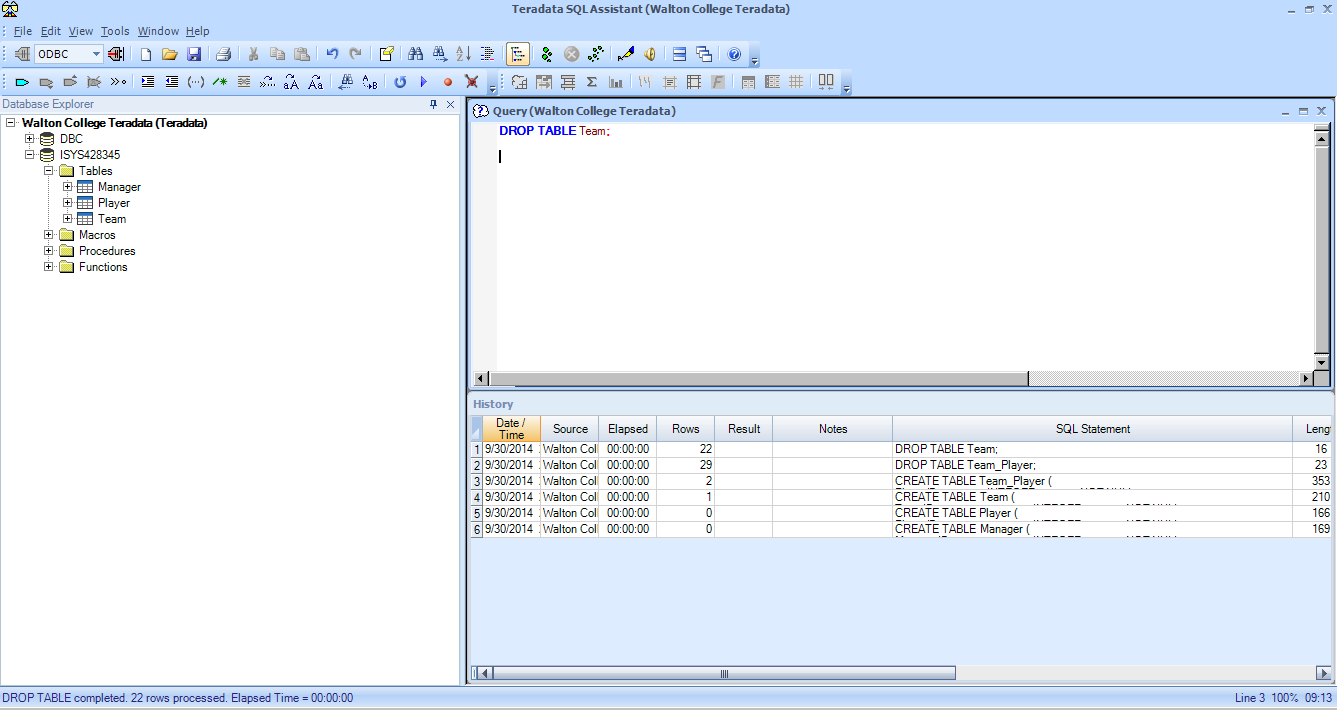
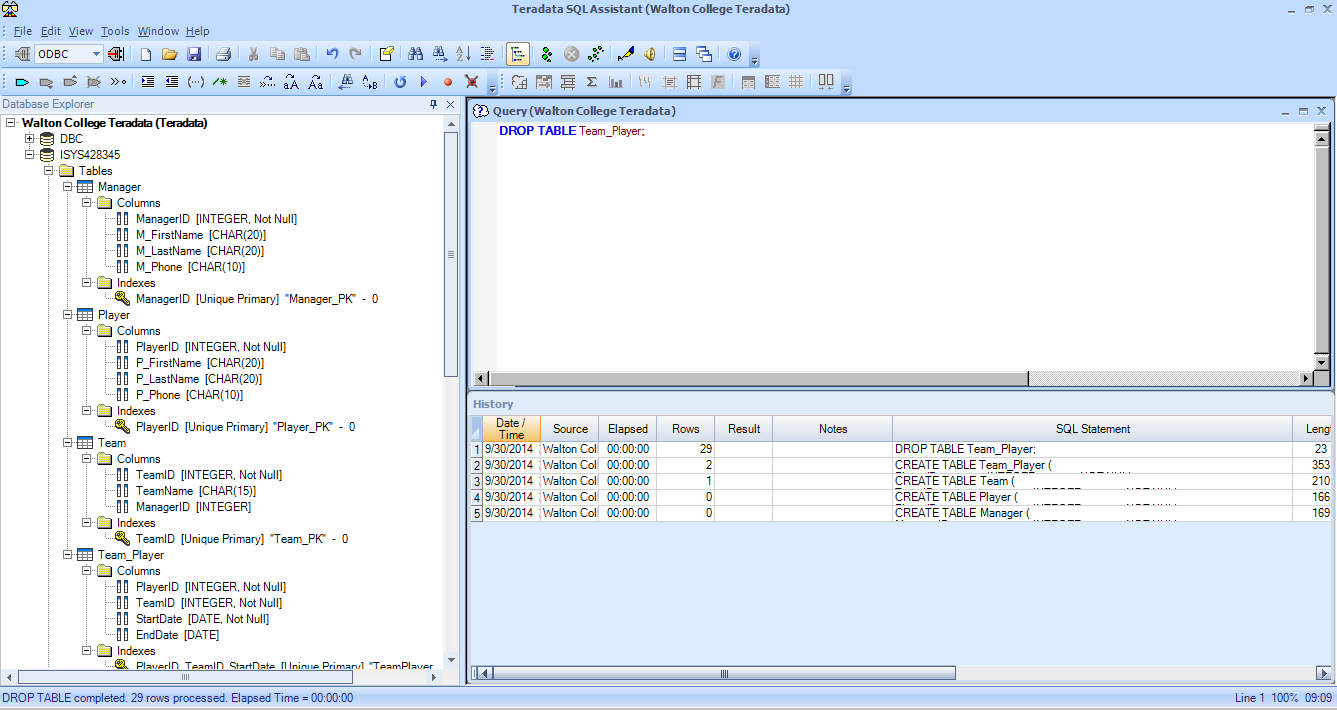
Two additional functions used within SQL queries are the CAST and TO\_DATE functions which allow you to convert data for displaying within the condition statement (WHERE).

* **TO\_DATE(**‘09/29/2015’**,’dd/mm/yyyy’) –** This converts text to a date.
* **CAST**('2015-09-29' **AS DATE)** – This also converts text to a date. Your date must be in a YYYY-MM-DD format.

# Part 9 – DROP & Delete Statements in SQL

As we discussed in our lecture, the use of the DROP and DELETE statements within SQL are very dangerous. It is very important to be careful when executing these statements.

The DROP statement within SQL allows us to remove our entire table from the DBMS schema. In class, we looked at the following examples of DROP statements.



In addition to this, we also discussed the DELETE statement in SQL. This allows you to delete the rows within your SQL table, not the structure itself. If I type **DELETE FROM** Customer; I am telling SQL to delete ALL rows within my Customer table. This is very dangerous.

You will typically have a WHERE clause within any of your DELETE statements to stipulate deleting only certain statements.   
**DELETE FROM** Customer   
**WHERE** CustomerState = ‘HI’;

# Part 10 – Your Turn – Writing SQL

Now it is your turn. You will login to SQL Server using the login information from class (also posted in the excel spreadsheet on Blackboard). It is highly recommended that you test all SQL statements on your database or the UA\_HALLUX database as instructed.

For your lab, type your answers into this word document. You can then submit your word document to the assignment area provided on Blackboard. After you’ve submitted your lab (and the deadline has passed), a solutions file will open up for you to review.

## **CREATE STATEMENTS**

**Question 1:**

Write the CREATE statements for the STUDENT, QUALIFIED, FACULTY, SECTION, COURSE, and REGISTRATION tables. Pick appropriate data types for the fields based on the example data given in the picture on the following page. Note: Remember your Relation shorthand. Refer back to Chapter 4 slides (STUDENT is your relation/table and StudentID and StudentName are the attributes/fields).

STUDENT (StudentID, StudentName)  
QUALIFIED (FacultyID, CourseID, DateQualified)  
FACULTY (FacultyID, FacultyName)  
SECTION (SectionNo, Semester, CourseID)  
COURSE (CourseID, CourseName)  
REGISTRATION (StudentID, SectionNo)

**Question 2:**

Does the order in which you run the create statements matter?

**Question 3:**

Are there any improvements that could be made to the design of these tables?

**Question 4:**

Write an INSERT statement to add a student with a student ID of 58745 and last name Smith to the Student table.

**Question 5:**

Write a statement that will remove this same student from Question 4 (only this student).

**Question 6:**

Write statements that will remove these 6 tables from your database (the entire table structure).

**Question 7**:

Does it matter in which order you remove these tables? Why or why not?

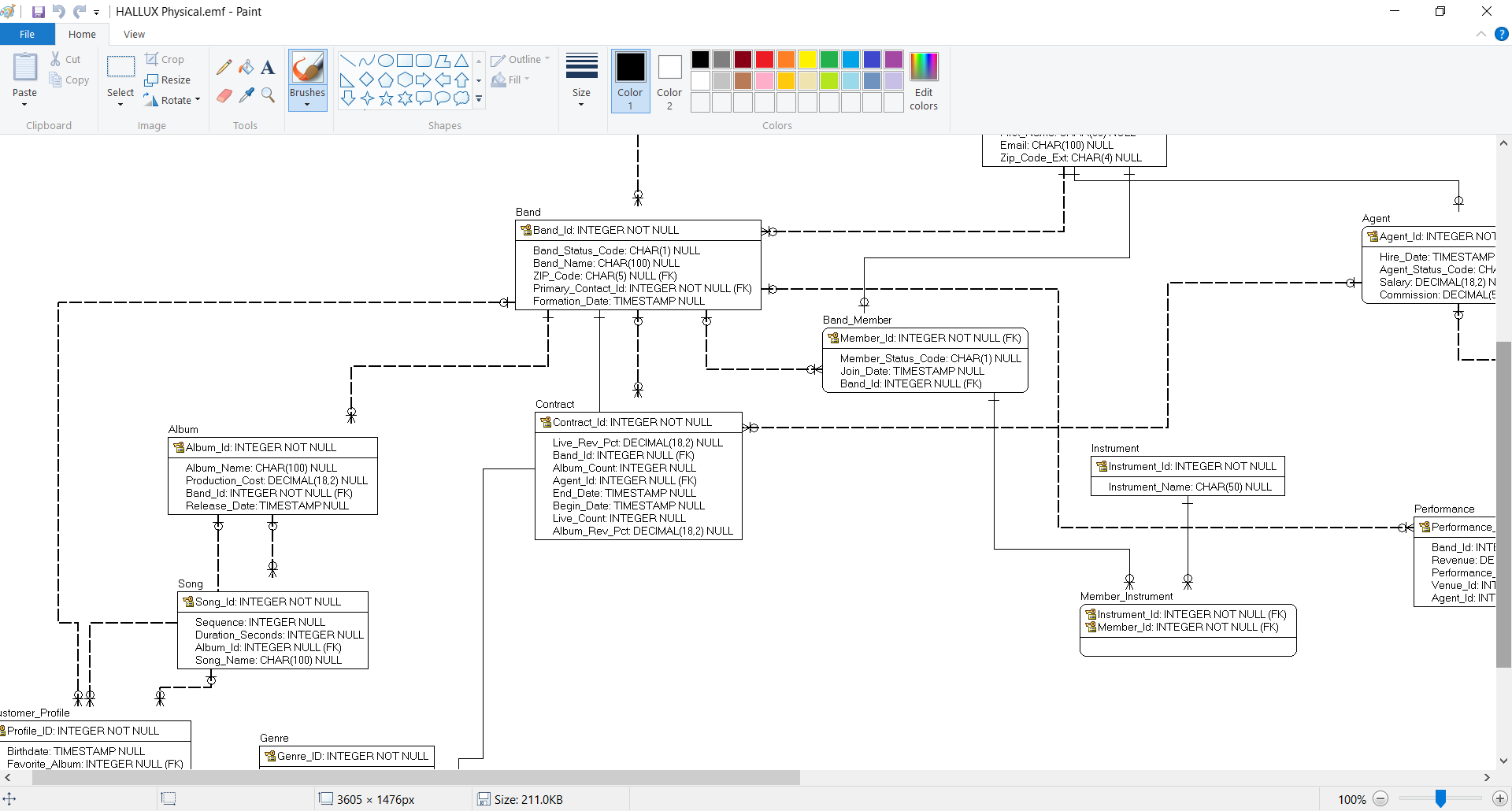


Use this image for the CREATE statements in question 1.

**Question 8:**

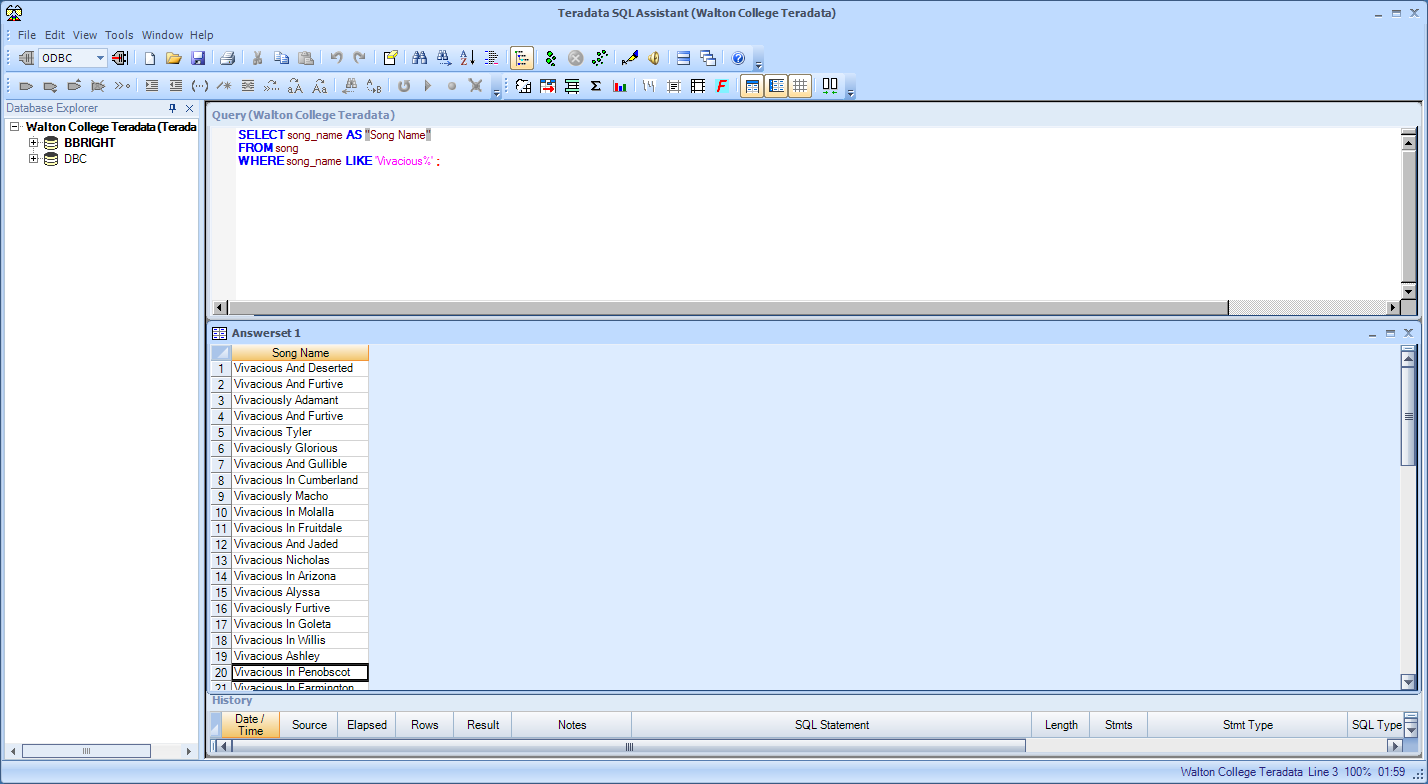
Execute the command required to use the UA\_HALLUX database for the following SELECT statements. What is the command required?

For the following statements, you will use the provided snippets of the UA\_HALLUX ERD to answer the questions below. The full revised ERD is too large to provide in this document and be easily read.



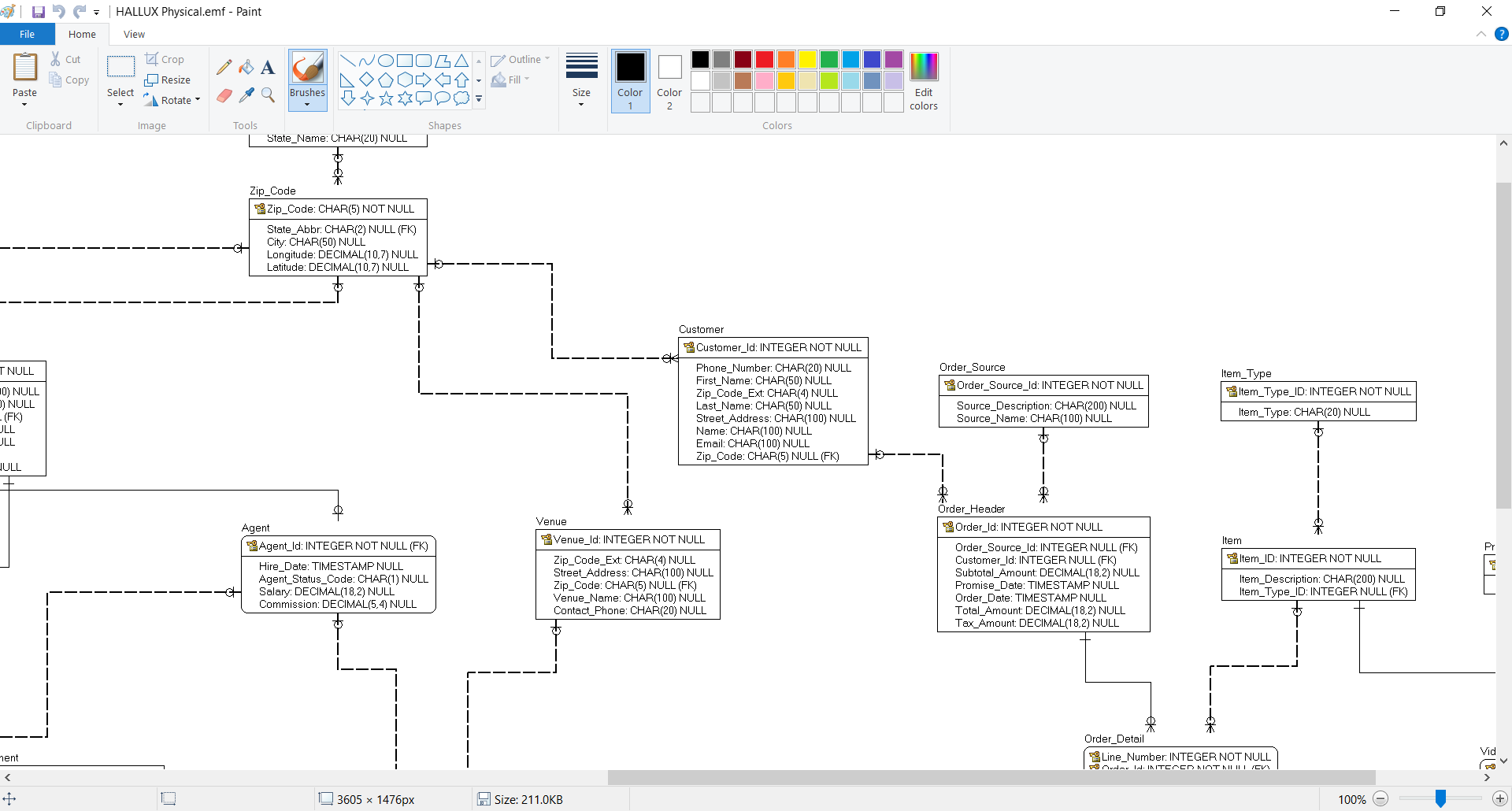
**Question 9:**

What songs begin with the word “Vivacious”? Use a column alias for formatting the column name.

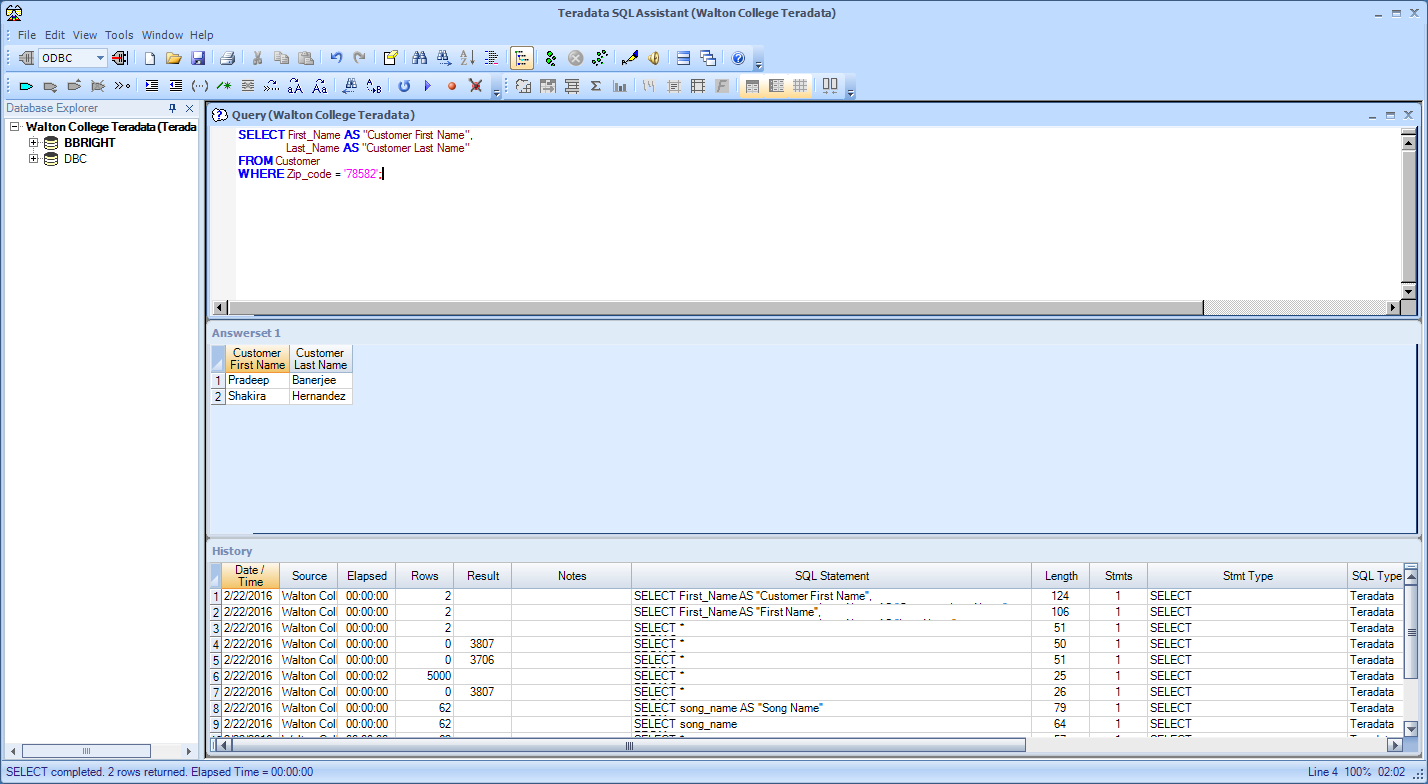


Your results should look like the image shown here:   
(You will have 62 results)

**Question 10:**

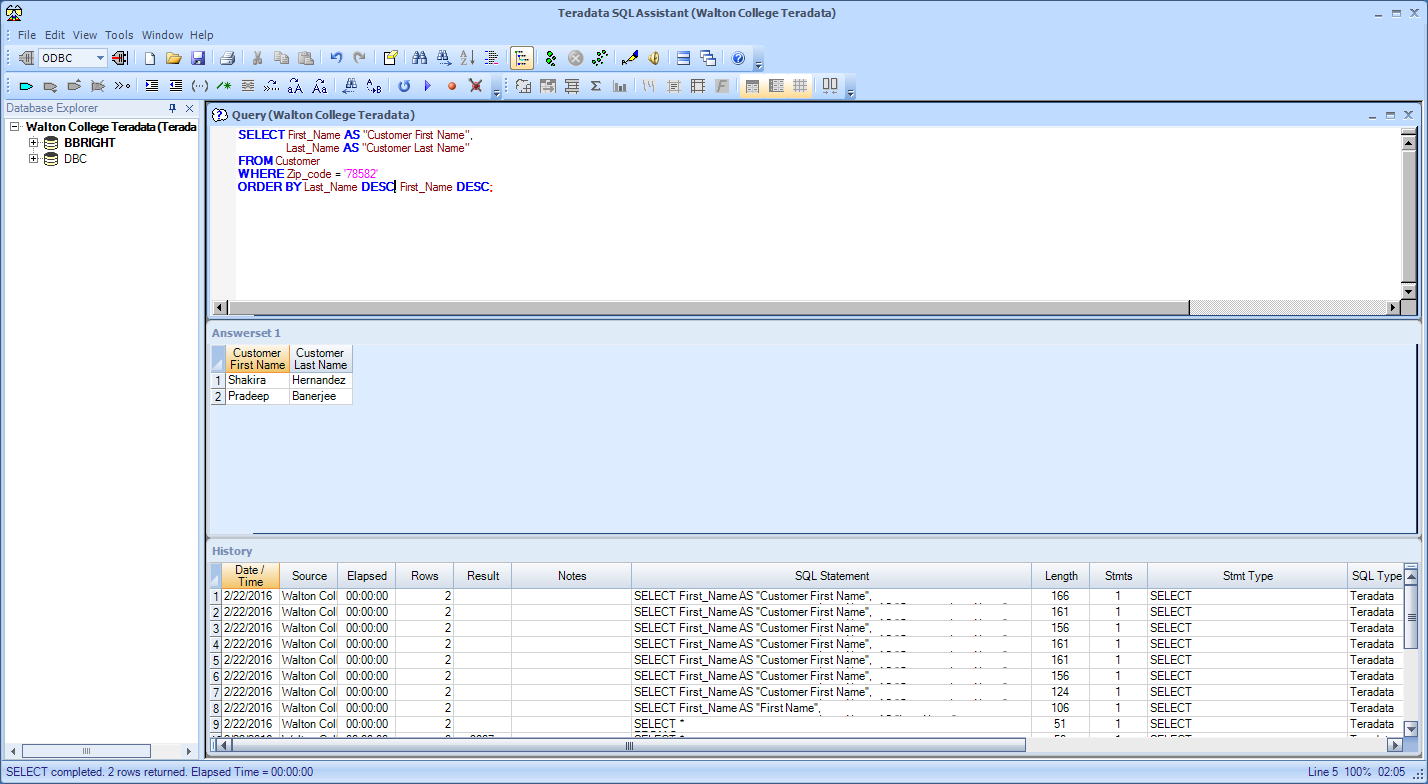


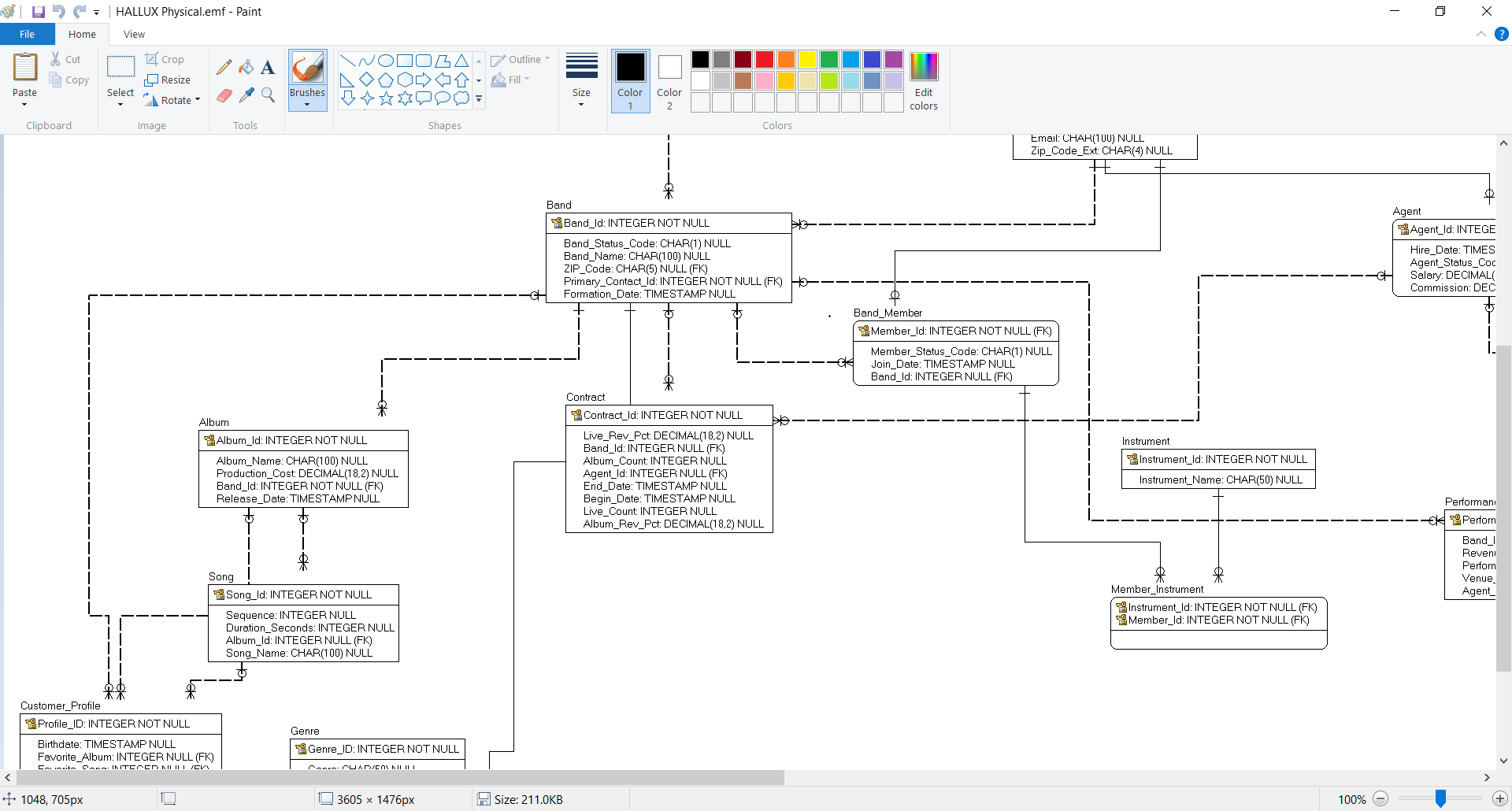
Who are the customers in zip code 78582? Display them in ascending order. Use a column alias for formatting column names.

Your results should be:

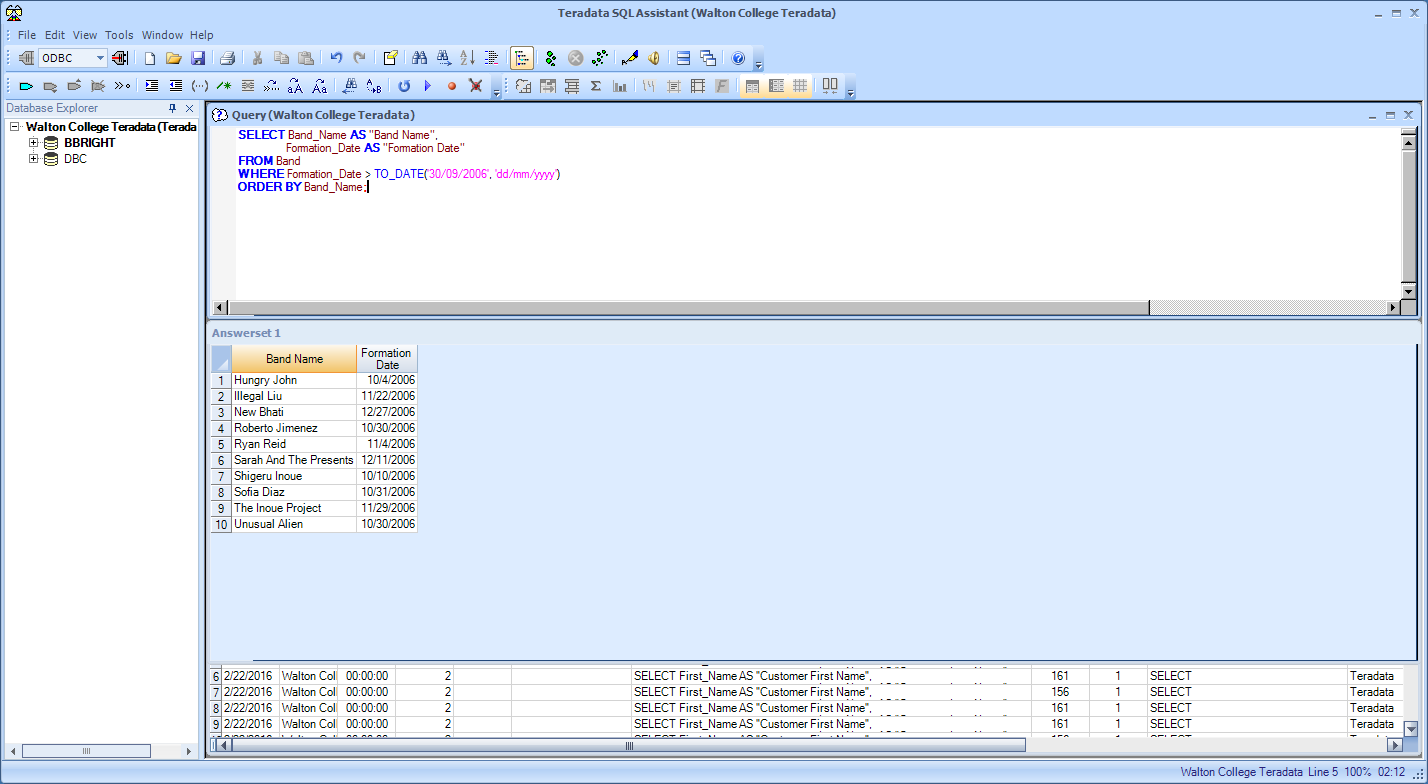
**Question 11:**

What changes do I make to my SQL to display these customers in descending order?

Your results should be:

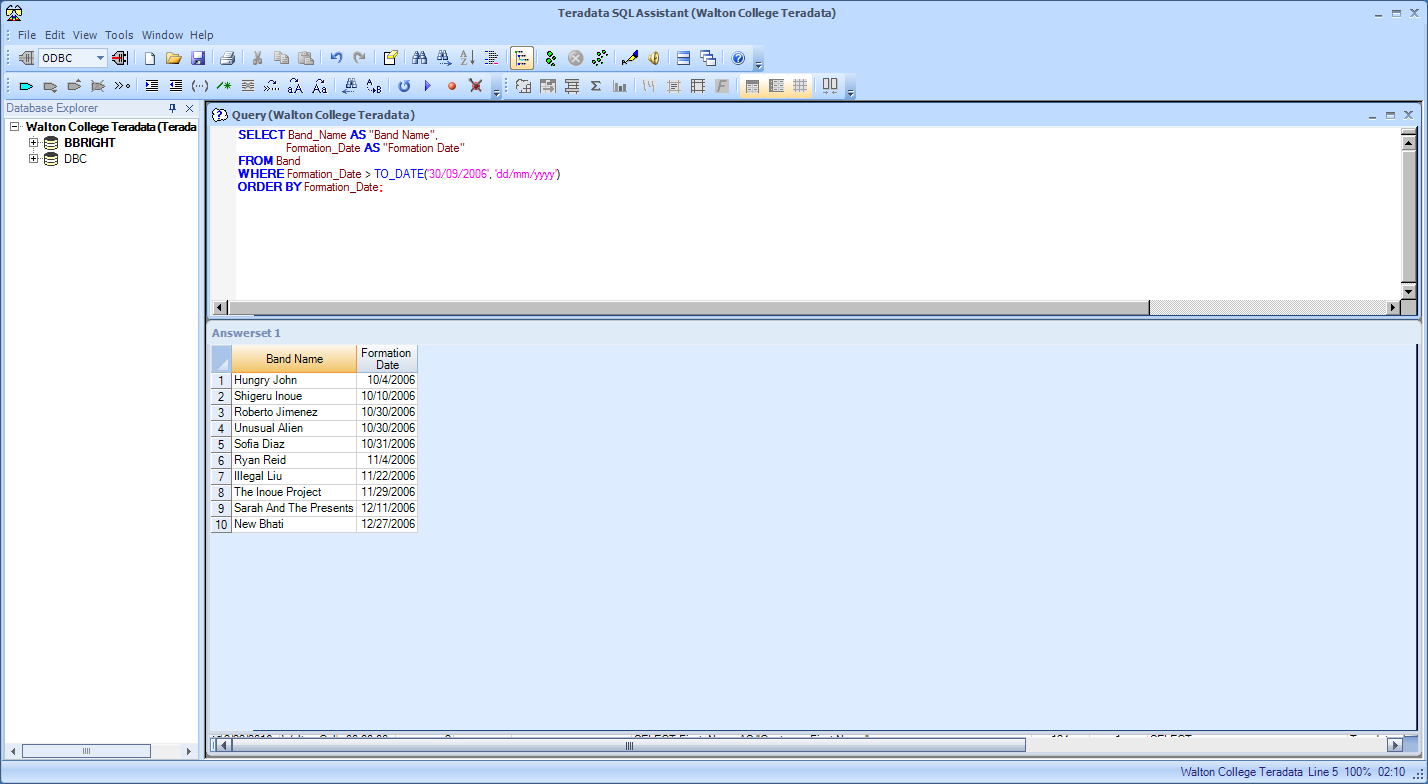
**Question 12:**

Your results should be:

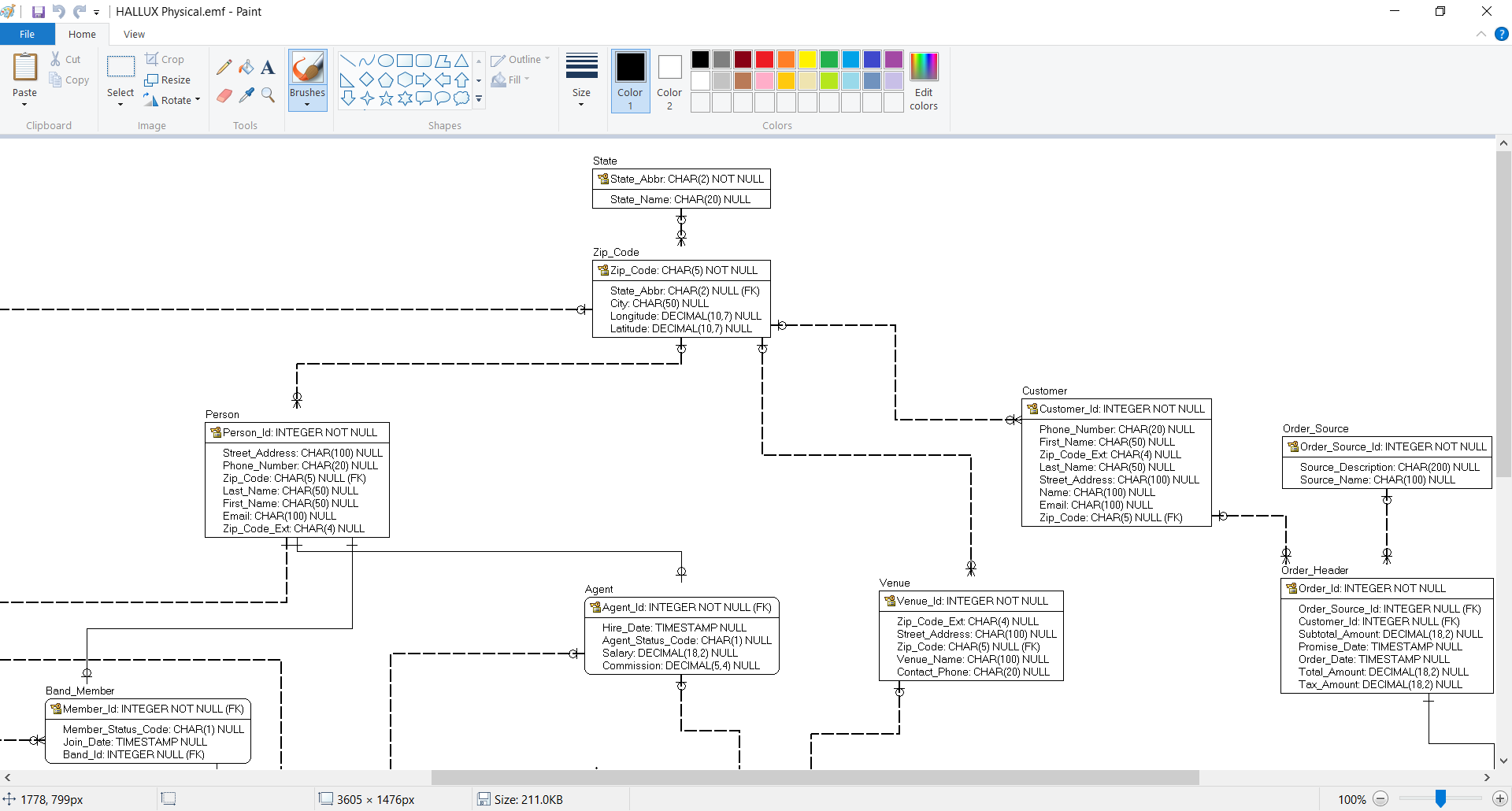
What bands were formed **after** September 2006? List them in order alphabetically*. (Hint: Refer back to the section above that looked at conversion functions).*

**Question 13:**

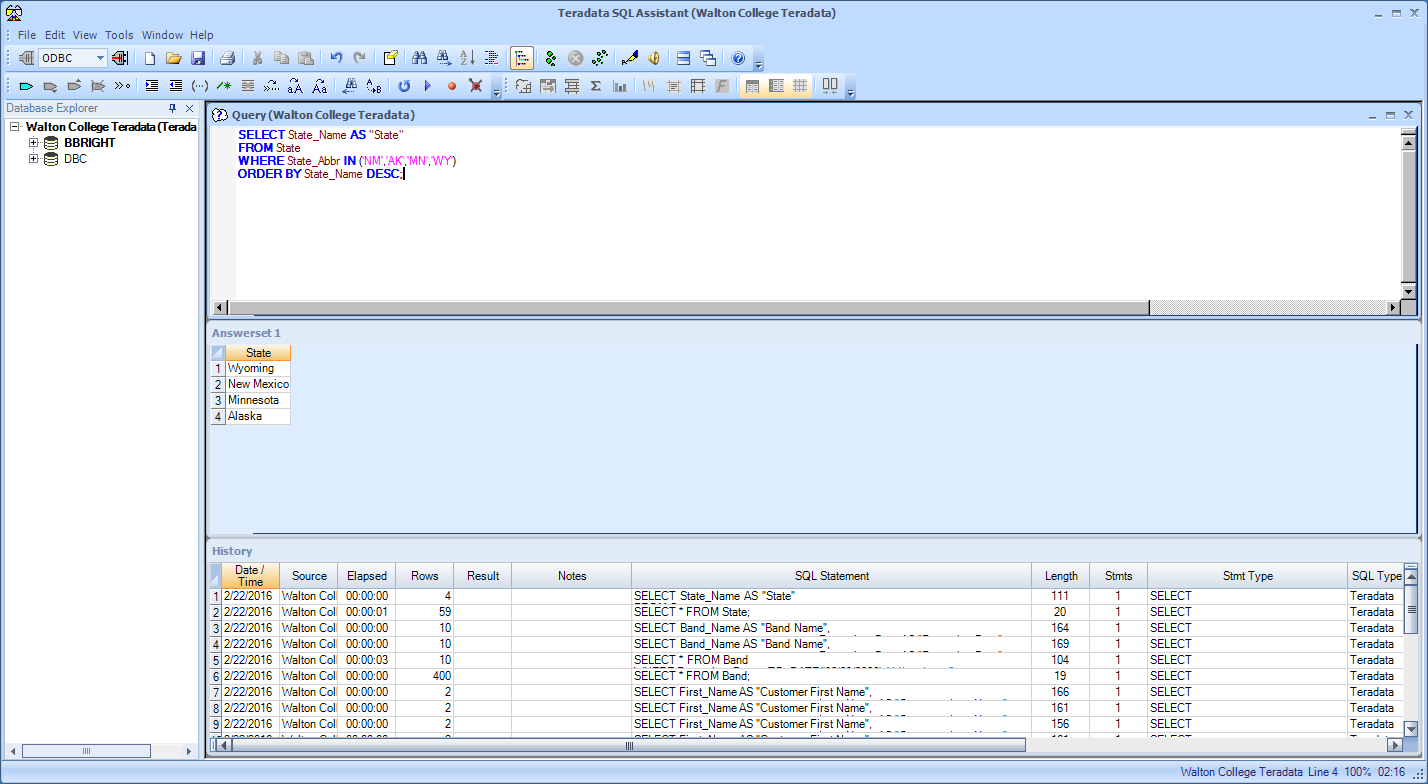
How would I change this query to display the bands in sequential order of their formation date?

Your results should be:

**Question 14:**

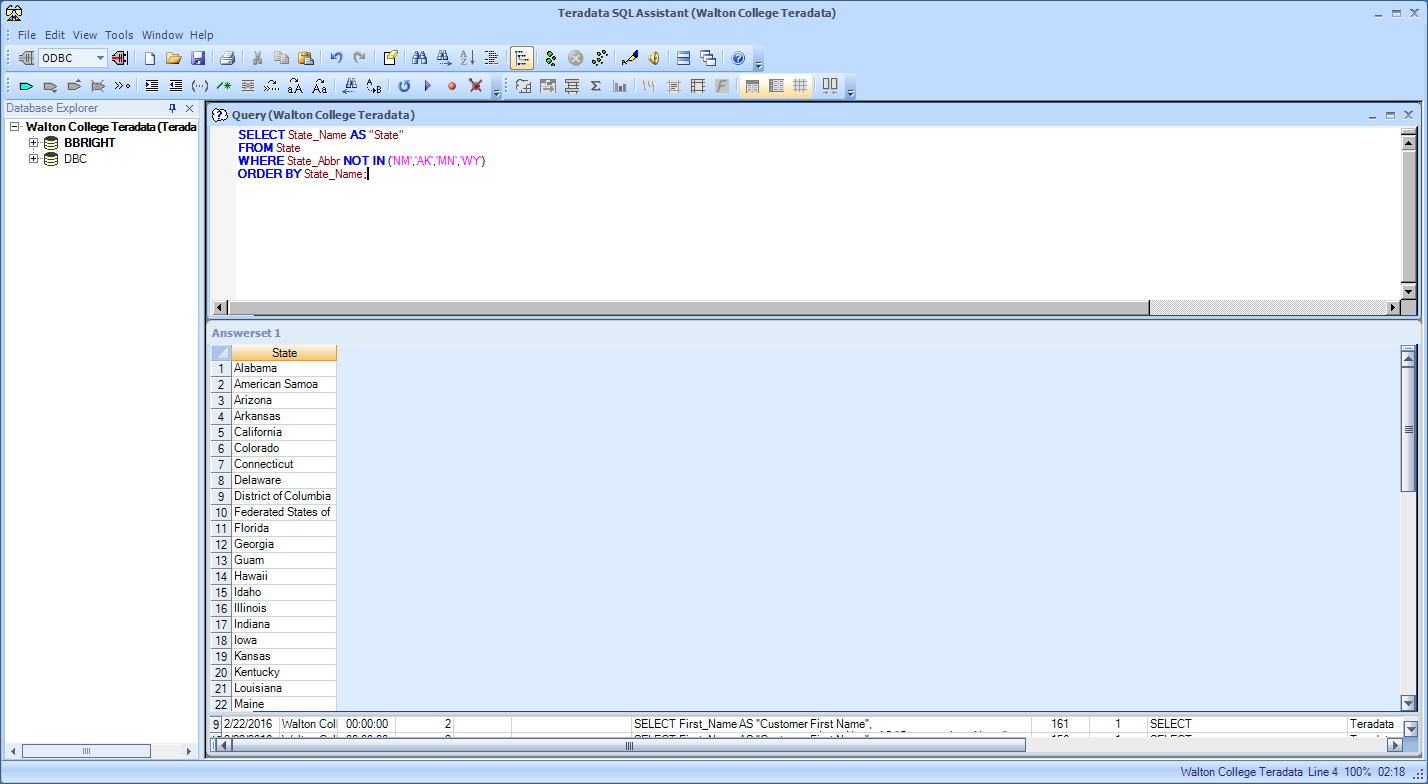


What are the state names for those with abbreviations of NM, AK, MN, and WY? List them in descending order.

Your results should be:

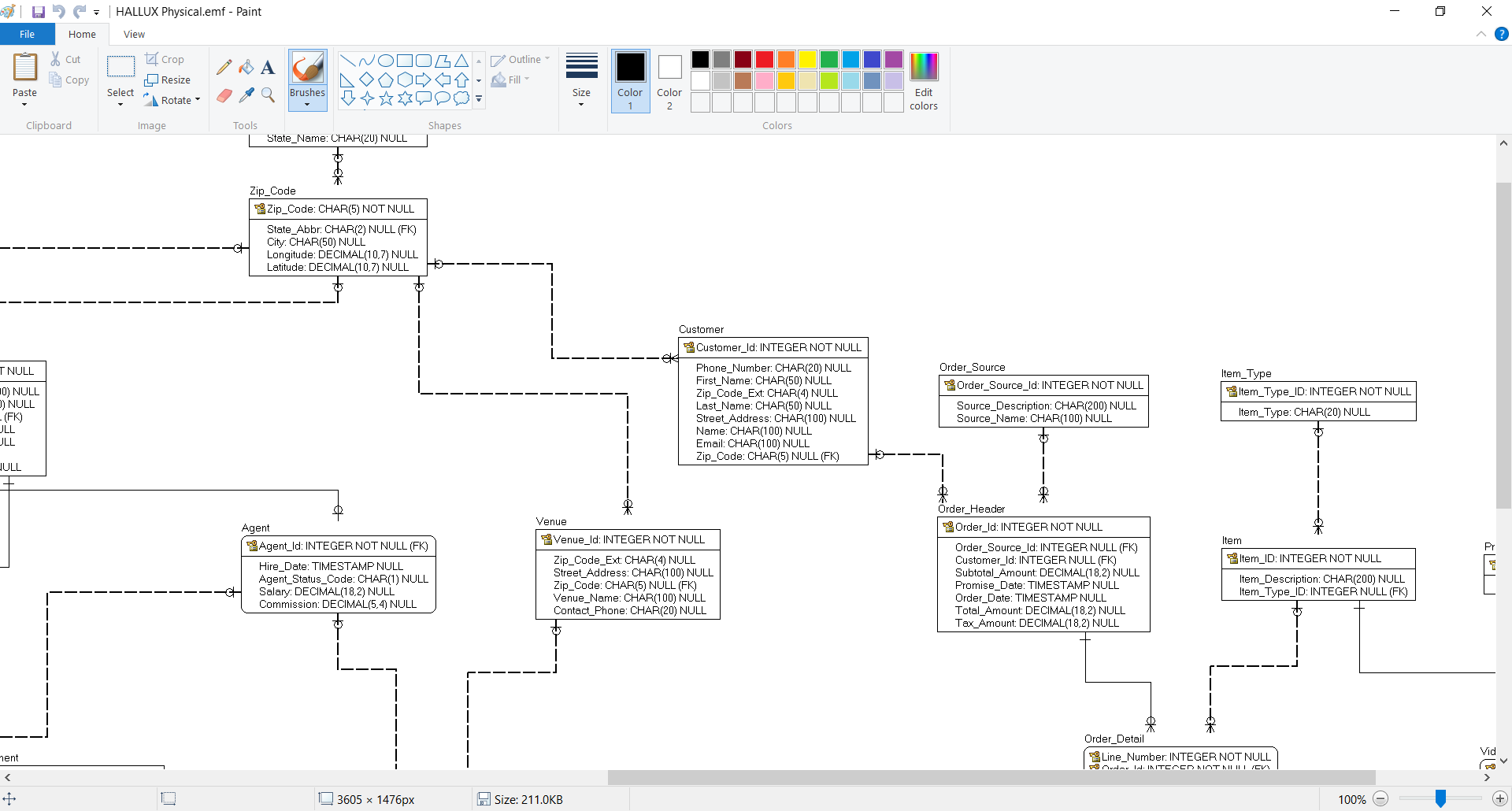
**Question 15:**

What are the state names for those with abbreviations **not** of NM, AK, MN, and WY? List them in ascending order. (Use the same ERD as Question 13)

Your results should be:

You should get 55 rows in your results, should **not** include the states listed in the question.

**Question 16:**



How many customers are there in each zip code? Show them in numerical order. Use an alias for the aggregate function column.

Your results should be:

You will have 4000+ rows (you will receive a warning telling you this – it won’t display all 4111).

**Question 17:**

Using your SELECT statement from question 16, only show the zip codes with more than 3 customers.

Your results should be:

You will have 22 rows, all counts should be greater than 3.

**Question 18:**

How would I display the results of question #17 in order of # of customers (the count)?

Your results should be:

