CS660 – DATABASE SYSTEMS

UNIT 3 - DATA DEFINITION AND DATA MANIPULATION LANGUAGE

Dr. John Conklin

Agenda

- SQL Language
- Writing SQL Statements
- Formulating SQL Statements
- Data Manipulation
- Data Definition
- Subqueries

SQL LANGUAGE

SQL Language

- A relational database's structured query language (SQL) is a computer language used to store and process data.
- In a relational database, data is stored in tabular form, with rows and columns representing various data attributes and the connections between the values of those attributes.
- To save, update, remove, search for, and retrieve data from the database, utilize SQL statements. SQL may also be used to enhance and maintain database performance.

WRITING SQL STATEMENTS

Writing SQL Statements

- Best practices for writing SQL queries:
  - **Follow the order of correctness, readability, and optimization** - here, the usual caution against premature optimization is applicable. Till you are certain that your SQL query produces the data you need, avoid tweaking it. Even then, only prioritize query optimization if it is used regularly.
  - **Reduce the size of your haystacks before looking for your needles** - It may be argued that we are already engaging in optimization at this point, but the objective should be to instruct the database to scan just the essential set of variables to return your findings. Interpretive design is one of SQL's many appealing features. You simply need to tell the database the records you need; the database will work out the most effective way to get those records without further instruction.
  - **Get to know your data first** - study the metadata to ensure that a column actually contains the data you anticipate before writing a single line of code to become familiar with your data.

▼ 1. Start your query with the select statement.

- **select** [all | distinct]
  A **select** statement queries the database and retrieves selected data that match the specified criteria.

▼ 2. Add field names you want to display.

- **field1** [,field2, 3, 4, etc.]
  The **field** name specifies the particular fields that contain the data.
  Note: You can find field and data table names in the Data Models section of DataLink.

▼ 3. Add your statement clause(s) or selection criteria.

- Required:
  - **from** table1[, table2]
    The **from** clause specifies the table(s) that contain the data.

  Optional:
- **where** conditions
  The **where** clause specifies the selection condition(s) by which data is retrieved.
- **order by** [column-list]
  The **order by** clause specifies the ordering or sorting of rows.
- **group by** [column-list]
  The **group by** clause groups the resulting rows in sets.
- **having** conditions
  The **having** clause allows a search condition and is used with the **group by** clause.

Image Source: https://blink.ucsd.edu/technology/help-desk/queries/SQL/index.html#4.-Review-your-select-statement

Writing SQL Statements

Review how to write a select SQL statement using this instruction.

FORMULATING SQL STATEMENTS

Formulating SQL Statements

- Structured Query Language, or SQL, is utilized by businesses when they have a lot of data they wish to modify (commonly pronounced like "sequel"). Anyone working for a company that keeps data in a relational database may use SQL.
- Transact-SQL (**T-SQL**) language is an extended implementation of the SQL for the Microsoft SQL Server.
- As the initial or origin of SQL queries, the **SELECT** statement may be thought of as their beginning point.
  - Data from the data tables is retrieved using the **SELECT** command.
  - The column names are initially specified in the **SELECT** query syntax, and a comma is used to separate them.
  - The **SELECT** statements don't contain any commas if only one column is used.
  - The **FROM** clause is written in the subsequent stage, and the table name is added as a final step.
  - The **SELECT** statement syntax for the example below, which pulls data from the Name and Surname columns, is as follows:
    - **SELECT** Name, SurName **FROM** Student (next slides shows this example)

Name	SurName	Lesson	Age	PassMark
Lawrence	Jerome	Ancient History	21	48.824
Jerome	Salvador	Roman History	20	18.382
Ernest	Nicholas	Roman History	19	32.587
Jorge	Gilbert	European History	19	81.11
Salvador	Ernest	Second World War History	20	43.425
Gilbert	Lawrence	European History	24	56.15
Nicholas	Jorge	Second World War History	22	29.685

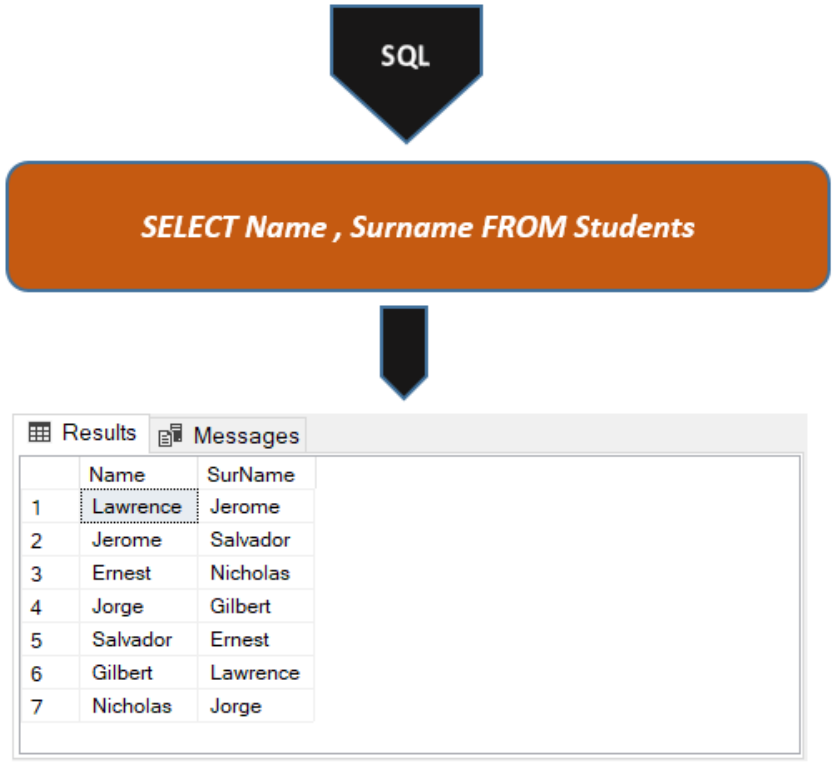


Image Source: https://www.sqlshack.com/learn-to-write-basic-sql-queries/

Formulating SQL Statements

This is the example SELECT statement from the previous slide.

Formulating SQL Statements

Results | Messages

	Name	SurName	Lesson	Age	PassMark
1	Lawrence	Jerome	Ancient History	21	48.824
2	Jerome	Salvador	Roman History	20	18.382
3	Ernest	Nicholas	Roman History	19	32.587
4	Jorge	Gilbert	European History	19	81.11
5	Salvador	Ernest	Second World War History	20	43.425
6	Gilbert	Lawrence	European History	24	56.15
7	Nicholas	Jorge	Second World War History	22	29.685

Image Source: https://www.sqlshack.com/learn-to-write-basic-sql-queries/

- All of the table's columns are defined by the asterisk (*****) symbol. The SELECT command returns every column from the Student database in the example on the left.
  - SELECT * FROM Student

Advice: Our primary goal should be to receive responses from SQL queries as quickly as feasible, with the least amount of resource use and shortest possible execution time. The asterisk (*) symbol must be avoided as much as possible when writing SELECT statements. This usage pattern results in increased IO, CPU, and network costs. As a result, we may stop using the asterisk sign and instead utilize only the essential columns in our queries if we don't need all of the table's columns.

Name	SurName	Lesson	Age	PassMark
Lawrence	Jerome	Ancient History	21	48.824
Jerome	Salvador	Roman History	20	18.382
Ernest	Nicholas	Roman History	19	32.587
Jorge	Gilbert	European History	19	81.11
Salvador	Ernest	Second World War History	20	43.425
Gilbert	Lawrence	European History	24	56.15
Nicholas	Jorge	Second World War History	22	29.685

SELECT * FROM Students WHERE Age >=20

Results | Messages

	Name	SurName	Lesson	Age	PassMark
1	Lawrence	Jerome	Ancient History	21	48.824
2	Jerome	Salvador	Roman History	20	18.382
3	Salvador	Ernest	Second World War History	20	43.425
4	Gilbert	Lawrence	European History	24	56.15
5	Nicholas	Jorge	Second World War History	22	29.685

Image Source: https://www.sqlshack.com/learn-to-write-basic-sql-queries/

Formulating SQL Statements - **WHERE**

Filtering the Data: **WHERE** clause

- The WHERE clause is used to filter the data based on predefined criteria. The filtering criteria must be specified following the WHERE clause. The pupils whose ages are more than or equal to 20 are found using the example below.

```
SELECT *
FROM  Student
WHERE Age >=20
```

Formulating SQL Statements - **LIKE**

- The logical operator **LIKE** enables the application of a unique filtering pattern to the **WHERE** condition in SQL queries. The primary wildcard to utilize in combination with the **LIKE** operator is the percentage symbol (**%**). The following search will return all of the students whose names begin with the letter J.

```
SELECT *
FROM  Student
WHERE Name LIKE 'J%'
```

Results | Messages

	Name	SurName	Lesson	Age	PassMark
1	Jerome	Salvador	Roman History	20	18.382
2	Jorge	Gilbert	European History	19	81.11

Image Source: https://www.sqlshack.com/learn-to-write-basic-sql-queries/

Formulating SQL Statements - **IN**

```
SELECT *
FROM  Student
WHERE Lesson IN ('Roman History','European History')
```

Results | Messages

	Name	SurName	Lesson	Age	PassMark
1	Jerome	Salvador	Roman History	20	18.382
2	Ernest	Nicholas	Roman History	19	32.587
3	Jorge	Gilbert	European History	19	81.11
4	Gilbert	Lawrence	European History	24	56.15

Image Source: https://www.sqlshack.com/learn-to-write-basic-sql-queries/

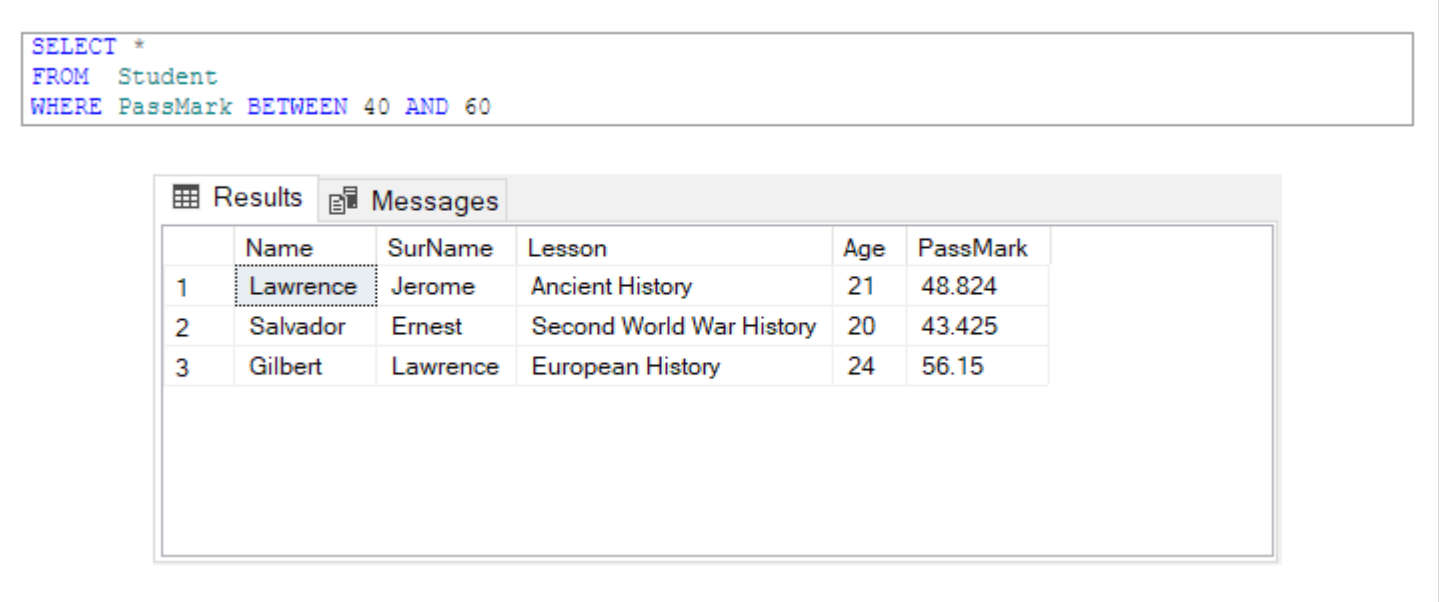
- We may apply several value filters to the WHERE clause using the **IN** operator. The following search retrieves information on the students who have taken Roman and European history classes.

```
SELECT *
FROM  Student
WHERE Lesson IN ('Roman History','European History')
```

Formulating SQL Statements - **BETWEEN**

- The data that fits inside the specified begin and finish values is filtered using the **BETWEEN** operator. The following search finds information for students with marks greater than 40 and equal to, but less than, 60.

```
SELECT *
FROM  Student
WHERE PassMark BETWEEN 40 AND 60
```



	Name	SurName	Lesson	Age	PassMark
1	Lawrence	Jerome	Ancient History	21	48.824
2	Salvador	Ernest	Second World War History	20	43.425
3	Gilbert	Lawrence	European History	24	56.15

Image Source: https://www.sqlshack.com/learn-to-write-basic-sql-queries/

Formulating SQL Statements – **ORDER BY**

```
SELECT *
FROM  Student
ORDER BY PassMark DESC
```

Results | Messages

	Name	SurName	Lesson	Age	PassMark
1	Jorge	Gilbert	European History	19	81.11
2	Gilbert	Lawrence	European History	24	56.15
3	Lawrence	Jerome	Ancient History	21	48.824
4	Salvador	Ernest	Second World War History	20	43.425
5	Ernest	Nicholas	Roman History	19	32.587
6	Nicholas	Jorge	Second World War History	22	29.685
7	Jerome	Salvador	Roman History	20	18.382

Image Source: https://www.sqlshack.com/learn-to-write-basic-sql-queries/

- We may sort the data using the **ORDER BY** command and the chosen column. You can choose to order the data's result set either ascendingly or descendingly. The keywords **ASC** and **DESC** are used to sort the data in ascending and descending order, respectively.
- The following query uses the PassMark column expressions to sort the student data in decreasing order.

```
SELECT *
FROM  Student
ORDER BY PassMark DESC
```

By default, **ORDER BY statement sorts data in ascending order.

Formulating SQL Statements - **DISTINCT**

- To ensure that the result set contains only unique (different) values, duplicate data is removed from the given columns using the distinctive clause.
- In the example that follows, we will obtain data from the Lesson column, but by using the **DISTINCT** clause, we will only return different values.

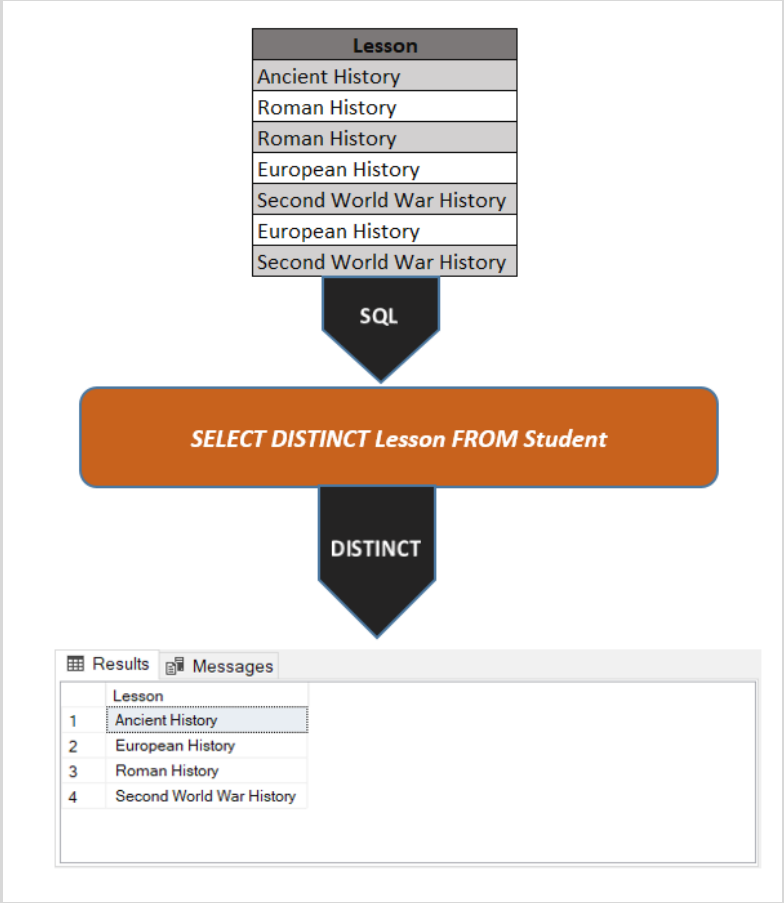


Image Source: https://www.sqlshack.com/learn-to-write-basic-sql-queries/

DATA MANIPULATION

Data Manipulation (DML)

- To query and alter database data, one uses the SQL data manipulation language (**DML**). The SELECT, INSERT, UPDATE, and DELETE SQL DML command statements are described below, and we will go through how to utilize them.

  - SELECT – to query data in the database
  - INSERT – to insert data into a table
  - UPDATE – to update data in a table
  - DELETE – to delete data from a table

Data Manipulation (DML)

- In the DML statement for SQL:
  - A statement's clauses should each start on a separate line.
  - Every clause's first word should match the first word of the preceding clause.
  - If a clause contains many parts, they should all be indented under the beginning of the sentence to indicate the link and appear on distinct lines.
  - Reserved words are represented with upper case letters.
  - User-defined terms are represented by lower case letters.

Data Manipulation (DML) - **SELECT**

The SELECT statement, or command, allows the user to **extract** data from tables, based on specific criteria.
It is processed according to the following sequence:

```
SELECT DISTINCT item(s)
FROM  table(s)
WHERE  predicate
GROUP BY  field(s)
ORDER BY fields
```

- Example:

```
SELECT  FirstName, LastName, phone
FROM Employees
ORDER BY LastName
```

Data Manipulation (DML) - **INSERT**

- The **INSERT** statement **adds rows** to a table. In addition,
  - **INSERT** specifies the table or view that data will be inserted into.
  - Column_list lists columns that will be affected by the INSERT.
  - If a column is omitted, each value must be provided.
  - If you are including columns, they can be listed in any order.
  - VALUES specifies the data that you want to insert into the table. VALUES is required.
  - Columns with the IDENTITY property should not be explicitly listed in the column_list or values_clause.

The syntax for the INSERT statement is:

```
INSERT [INTO] Table_name | view name [column_list]
DEFAULT VALUES | values_list | select statement
```

Data Manipulation (DML) - **UPDATE**

- The **UPDATE** statement changes data in existing rows either by adding new data or modifying existing data.

- The syntax for the UPDATE statement is:

```
UPDATE [Table Name]
SET [Field] = [Value]
WHERE [clause]
```

Data Manipulation (DML) - **DELETE**

- A record set's rows are deleted using the **DELETE** command. Only one table or row may be shown at a time when using the **DELETE** command to delete rows from a table or view. **WHERE** restricts the deletion to particular entries using the standard **WHERE** clause.

- The DELETE syntax looks like this.

  DELETE [FROM] {table_name | view_name }
  [WHERE clause]

Data Manipulation (DML) - **Aggregate functions**

FUNCTION	DESCRIPTION
AVG	Returns the average of all the values, or only the DISTINCT values, in the expression.
COUNT	Returns the number of non-null values in the expression. When DISTINCT is specified, COUNT finds the number of unique non-null values.
COUNT(*)	Returns the number of rows. COUNT(*) takes no parameters and cannot be used with DISTINCT.
MAX	Returns the maximum value in the expression. MAX can be used with numeric, character and datetime columns, but not with bit columns. With character columns, MAX finds the highest value in the collating sequence. MAX ignores any null values.
MIN	Returns the minimum value in the expression. MIN can be used with numeric, character and datetime columns, but not with bit columns. With character columns, MIN finds the value that is lowest in the sort sequence. MIN ignores any null values.
SUM	Returns the sum of all the values, or only the DISTINCT values, in the expression. SUM can be used with numeric columns only.

Image Source: https://opentextbc.ca/dbdesign01/chapter/chapter-sql-dml/

- Aggregate functions perform a calculation on a set of values and return a single, or summary, value.

DATA DEFINITION

Data Definition (DDL)

- The section of SQL that creates, modifies, and deletes database objects is known as the data definition language (**DDL**).
- Schemas, tables, views, sequences, catalogs, indexes, variables, masks, permissions, and aliases are some examples of these database objects.
- To define the database structures, such as to specify the relations or tables for a database along with their properties, data definition language commands are employed.
  - **CREATE**: To create a new relation in a database, use the CREATE command.
  - **DROP**: To delete or get rid of an existing relation in the database, use the DROP command.
  - **ALTER**: To change an existing relation in a database, use the ALTER command.
  - **TRUNCATE**: The TRUNCATE command is used to remove every instance of the table, which keeps the table's or relation's outer structure intact.
  - **RENAME**: The ALTER command is used in conjunction with the RENAME command to change a relation's name or one of its attributes.

Data Definition (DDL) – **CREATE TABLE**

- To create a table, you use a CREATE TABLE command.

- A CREATE TABLE command has the following syntax:

```
CREATE TABLE table_name
 (field1 data type [(size)] [NOT NULL] [index1]
 [, field2 data type [(size)] [NOT NULL] [index2]
 [, ...][, CONSTRAINT constraint1 [, ...]])
```

Data Definition (DDL)

- Character strings, also referred to as **VARCHAR** or **CHAR** for variable or fixed length strings.
- Numeric types like **NUMBER** or **INTEGER**, which will typically define a precision, and
- **DATE** or related kinds, are the data types that you will use the most frequently.
- Data type syntax varies from system to system; the only way to be certain is to look in the manual for the software you are using.

SUBQUREIES

Subqueries

- A subquery is a SQL query nested inside a larger query.
  - A subquery may occur in :
    - - A SELECT clause
    - - A FROM clause
    - - A WHERE clause
  - The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.
  - A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
  - You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.
  - A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.
  - The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

Syntax :

```
SELECT    select_list
FROM      table
WHERE     expr operator
                    (SELECT    select_list
                     FROM      table);
```

- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the subquery result.

Image Source: https://www.w3resource.com/sql/subqueries/understanding-sql-subqueries.php

Subqueries

You can use a subquery in a SELECT, INSERT, DELETE, or UPDATE statement to perform the following tasks:

- Compare an expression to the result of the query.
- Determine if an expression is included in the results of the query.
- Check whether the query selects any rows.

Subqueries

- There are some guidelines to consider when using subqueries :
  - A subquery must be enclosed in parentheses.
  - A subquery must be placed on the right side of the comparison operator.
  - Subqueries cannot manipulate their results internally, therefore **ORDER BY** clause cannot be added into a subquery. You can use an **ORDER BY** clause in the main **SELECT** statement (outer query) which will be the last clause.
  - Use single-row operators with single-row subqueries.
  - If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a **WHERE** clause.

INDIVIDUAL PROJECT

Individual Project

Description

- The case study retail store has provided a list of reports and data manipulation tasks that are needed in the processing of orders for their customers. Answer the following:
- What structured query language (SQL) statement scripts are needed to create the database schema for the relational database system and manipulate the data in the solution that you are proposing to the company?
- How does each of these scripts specifically support the goals and objectives of the company?

Individual Project

- The project deliverables are as follows:
  - **Data Manipulation Tasks**
    - Insert 20 records into each table for testing purposes.
    - Delete an entire order by using the unique identifier for that order.
    - Update the price of a product by using the unique identifier for that product.
    - Add a minimum of 3 of your own data manipulation language (DML) scripts based on the needs and specifications of your retail store.
  - **Report List**
    - Total revenue (sales) per month, grouped by customer
    - Total revenue (sales) per month, grouped by product
    - Total count of products, grouped by category
    - Add minimum of 3 of your own report scripts based on the needs and specifications of your retail store (one must be a CROSSTAB)

Individual Project

SQL (4–5 pages)

- Include the database definition language (**DDL**) scripts to **CREATE** to database schema as described in the entity–relationship (E–R) diagram (Unit 2).
- Include the database manipulation scripts (**DML**) that will be used to **INSERT**, **DELETE**, and **UPDATE** data in the proposed database system.
- Include the **SELECT**, **CROSSTAB**, and **AGGREGATE FUNCTION** statements that will be used to read data from the proposed database system.
- Provide your analysis as to how this part of the project fulfills the mission and 1 or more goals of the case study organization.
- Provide the following attachments (in **addition to embedding** in document):
  - DDL.sql (including CREATE and INSERT statements so that they execute in the correct order [top-down])
  - DML.sql (including DELETE and UPDATE statements so that they can be executed in any order as selected)
  - REPORT.sql (including SELECT, CROSSTAB, AGGREGATE FUNCTION statements so that they can be executed in any order as selected)
- Note: **You will embed each script in the Word document and also <u>provide it as an attachment</u>**.
- All sources should be cited both in-text and in References using APA format.
- Name the document "**yourname_CS660_IP3**.doc."

Contact Information

Email:	Jconklin@coloradotech.edu
Phone:	602.796.5972
Website:	http://drjconklin.com
Office Hours:	Wednesdays: 6:00 PM – 7:00 PM (CST)
	Saturdays: 11:00 AM – 12:00 PM (CST)
Live Chats:	Wednesdays: 7:00 PM – 8:00 PM (CST)