

WEEK #1

ASSIGNMENT

2% Group Assignment

Introduction to Databases

Mark Morell

Database Management – Fall 2019

Assignment Type:

- Groups of 2 to 4 students (please list all participants)

Date Due:

- Thursday, September 12th by start of class

Instructions:

- Please submit your assignment electronically through the eConestoga portal.
- Assignments should be submitted as Microsoft Word files.
- Include a separate title page as part of your submission that includes your name as well as your group member names. Everyone in the group should must submit an identical copy of the assignment.
- If you are using external sources (images, text, etc.) you must reference them as part of your assignment and not copy them as-is.
- Best practice is to research your answers and then write the response to the question in your own words.
- Your responses to the assignment should be well thought-out, complete and clear to the reader.
- Please include the question number with your responses.

Late Assignments:

Days Late	Penalty %
1	5
2	10
3	20
4	40
5	60
6	80
7	100

Assignment Questions

Question #	Question	Score
1	Identify 4 different Relational Database Management Systems (RDBMS) and describe some of the differences between them	4
2	Research the popularity of various RDBMS vendors among both small and large businesses and demonstrate how much each is being used. You must note the source of your data.	2
3	Discuss the differences between relational databases and non-relational databases and highlight what situations are best for each	3
4	Explain what a “Data Warehouse” is as if you were telling someone who knows nothing about databases. Describe its characteristics as well as why you would use one.	4
	Assignment Format	2
	Total	15



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE®

Database Management (PROG8080)

Fall 2019 - Week 1 – Thursday Lesson



Agenda

- Welcome and Introductions
- Contact Information / Resources
- Expectations
- Course Topic Introduction
- Terminology / Acronyms / Resources
- Microsoft SQL Server Installation
- Assignment & Summary



Welcome!

- Professor/Instructor Mark Morell
 - mmorell@conestogac.on.ca
- Can also contact through eConestoga course email
- No current office hours but please contact if you have questions or need help to arrange a time to meet



Resources

- eConestoga
 - Course Outline
 - Instructional Plan
 - Weekly Lesson Material
 - Assignments
 - Grades
 - Additional Resources
 - Announcements



Expectations

- **DO** come to class and bring your laptop
- **DO** respect each other and the classroom
- **DO** be *open* to new ideas and ways of learning
- **DO** *participate* in class (with a “pass” option)
- **DO** help make class a positive environment
- **DO** your assignments on time
- **DO** check eConestoga regularly
- **DO** contact me for questions / help
- **DO** let me help you succeed



Classroom Rules

1. Please don't talk while the teacher is addressing the class
2. Please don't talk while another student is addressing the teacher or class
3. When students are working on a task, their conversation should relate to the task
4. If a student enters the classroom while the lesson is in progress, please don't let it disrupt the class



Quick Class Evaluation

- NOT graded
 - Just gives me some insight into where everyone is at in their database knowledge
- Go to www.menti.com (laptop or phone) and enter the code that appears at the top of the screen



Fun Facts

More than 10 people a year are killed by a vending machine

Summer on Uranus lasts 21 years

- 0.3% of solar energy from the Sahara is enough to power the whole of Europe
- One human hair can support 3 ounces
- There are 635,013,559,599 possible hands in a game of bridge.



Topic Introduction

- Data
 - Types of data
 - Store in a “Database”
 - An organized collection of data
- Manage with a Database Management System (DBMS)
 - Relational DBMS (RDBMS) vs Non-Relational DBMS (NoSQL)



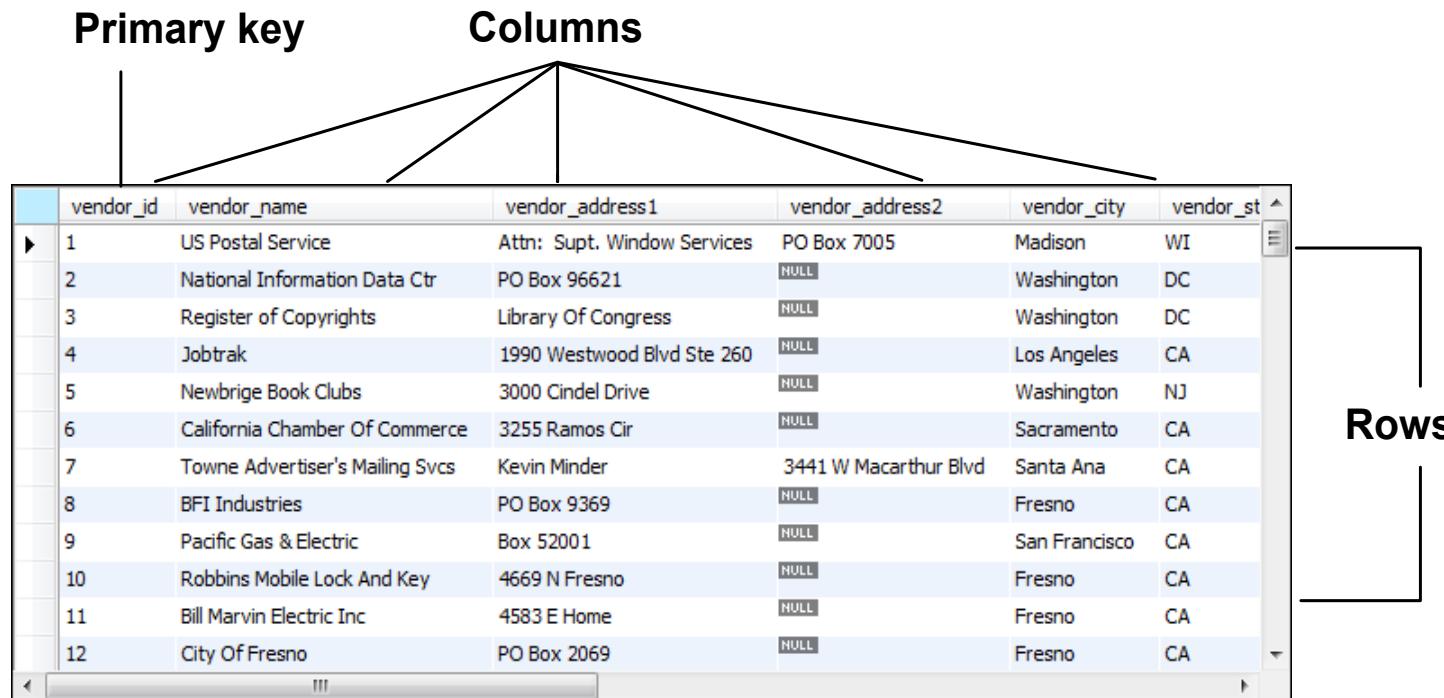
Topic Introduction Part 2

- RDBMS
 - Many types (Oracle, MySQL, MS SQL Server, etc.)
 - Why use one? Maybe you have/need:
 - Important data of varying sizes and types
 - Multiple users accessing data at the same time
 - Need a robust environment that provides good performance, security and ensures data integrity
 - Provide tools/services (reporting, integration, Business Intelligence management)



Topic Introduction Part 3

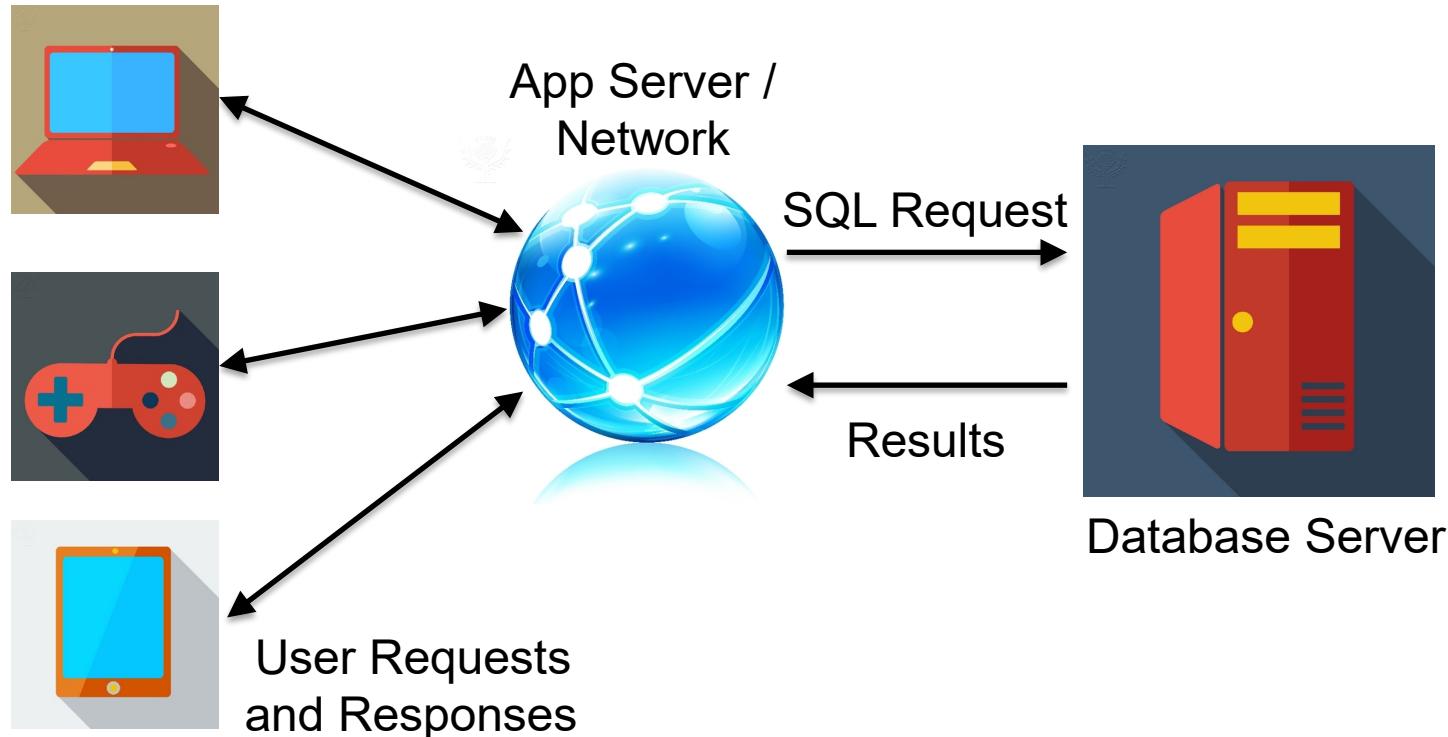
- Data in RDBMS Organized as Tables





Topic Introduction Part 4

- The Architectural “Fit”



Topic Introduction Part 5

- How to access/manipulate the data?
 - SQL (Structured Query Language)
 - Different RDBMS platforms have their own extensions to SQL:
 - Microsoft/Sybase - Transact-SQL (T-SQL)
 - Oracle - Programming Language for SQL (PL/SQL)
 - Tools provided by vendors
 - MS SQL Server uses SQL Server Management Studio (SSMS)



Terminology / Acronyms

- Data
- Database (DB)
- Database Management System (DBMS)
- Relational DBMS (RDBMS)
- SQL Server
- SSMS
- SQL
- Transact-SQL (T-SQL)





Valuable Resources

Microsoft SQL
Server
Documentation

W3schools

Stack Overflow

Lynda
(LinkedIn
Learning)

YouTube



Microsoft SQL Server

- We will be using SQL Server 2017
 - “Developer” Version
 - Full-featured (Don’t download the “trial”)
 - Licensed for use in a non-Production environment
 - Download the free “Developer” Edition here:
[Microsoft SQL Server 2017 Download](#)
 - Install it on your laptop after downloaded
 - The “Basic” Installation Type Option is all that we need



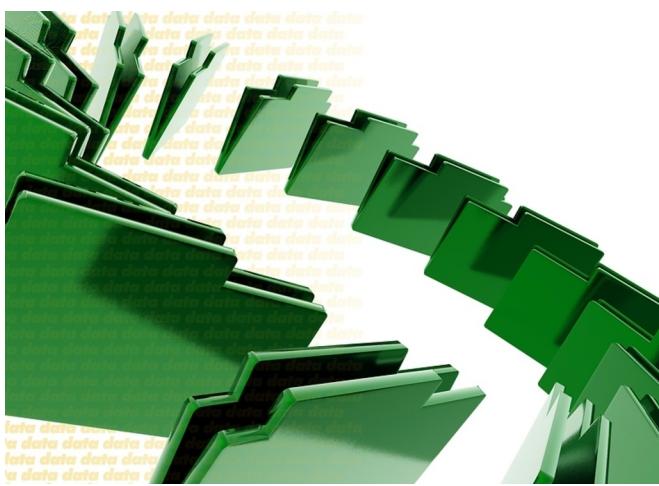
Microsoft SQL Server - Studio

- SQL Server Management Studio (SSMS)
 - Will be prompted to install after SQL Server is completely set up
 - Or find it here:
[SQL Server Management Studio Download](#)
 - Install it to your laptop after download
 - You may need to reboot and re-install if certain used components don't exist on your laptop
 - Launch the application, “Connect” (log in) using Windows credentials and look around



Summary / Assignment

- High-Level Overview & Concepts
- Terminology
- SQL Server 2017 Installation
- Assignment #1 (Groups of 2 – 4)
 - Find it and submit it in eConestoga
 - CAREFULLY read the directions in the assignment
 - Due by class next Thursday (September 12th)
- Questions?



WEEK #2

ASSIGNMENT

2% Group Assignment

Databases Basics

Mark Morell

Database Management – Fall 2019

Assignment Type:

- Groups of 2 to 5 students (please list all participants)

Date Due:

- Thursday, September 19th by the end of the day

Instructions:

- Please submit your assignment electronically through eConestoga.
- Assignments should be submitted as Microsoft Word files using the course coversheet format.
- Ensure that everyone from your group is listed on the coversheet of the assignment. Everyone in the group should must submit an identical copy.
- If you are using external sources (images, text, etc.) you must reference them as part of your assignment and not copy them as-is.
- Best practice is to research your answers and then write the response to the question in your own words.
- Please include the question number with your responses.
- For SQL results please include print screens of the SQL used as well as the full results of the query.

Late Assignment Penalty:

Days Late	Penalty %
1	5
2	10
3	20
4	40
5	60
6	80
7	100

Assignment Questions

Question #	Question	Score																									
1	<p>For each of the following types of data state which T-SQL Data Type would be most appropriate and <i>briefly</i> describe why you would choose it:</p> <ul style="list-style-type: none"> a) A list of street addresses for businesses b) A picture from Twitter c) A list of car license plate numbers d) A list of holidays e) A list of prices for movie rentals f) The answer to the question “Did you eat lunch today?” 	3																									
2	<p>What is the precision and scale of the following numbers:</p> <ul style="list-style-type: none"> a) 22573.41 b) 527 <p>What is the length of the following strings (between the quotation marks):</p> <ul style="list-style-type: none"> a) “I’ll_Be_Back!” b) “Friends don’t let friends drink and drive” 	2																									
3	<p>Answer these questions using the following “Hockey Players” table:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>hockeyID</th> <th>jerseyNumber</th> <th>mainTeam</th> <th>fullName</th> <th>birthDate</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>99</td> <td>Edmonton</td> <td>Wayne Gretzky</td> <td>26/01/1961</td> </tr> <tr> <td>2</td> <td>4</td> <td>Boston</td> <td>Bobby Orr</td> <td>20/03/1948</td> </tr> <tr> <td>3</td> <td>9</td> <td>Detroit</td> <td>Gordie Howe</td> <td>31/03/1928</td> </tr> <tr> <td>4</td> <td>66</td> <td>Pittsburgh</td> <td>Mario Lemieux</td> <td>05/10/1965</td> </tr> </tbody> </table> <ul style="list-style-type: none"> a) What are the various candidate keys for this table? b) Are each of these keys natural keys or surrogate keys? c) What would make the best choice for the primary key and why? 	hockeyID	jerseyNumber	mainTeam	fullName	birthDate	1	99	Edmonton	Wayne Gretzky	26/01/1961	2	4	Boston	Bobby Orr	20/03/1948	3	9	Detroit	Gordie Howe	31/03/1928	4	66	Pittsburgh	Mario Lemieux	05/10/1965	3
hockeyID	jerseyNumber	mainTeam	fullName	birthDate																							
1	99	Edmonton	Wayne Gretzky	26/01/1961																							
2	4	Boston	Bobby Orr	20/03/1948																							
3	9	Detroit	Gordie Howe	31/03/1928																							
4	66	Pittsburgh	Mario Lemieux	05/10/1965																							
4	<p>Using the SIS database, write a query to retrieve the following from the Course table:</p> <ul style="list-style-type: none"> - Course Number - Number of Hours - Course Name (English) <p>Sort the results in descending order by course name and provide both a screenshot of the query you used and a screenshot of the resulting data.</p>	2																									
5	<p>Using the SIS database, write a query to retrieve just the unique country codes from the Person table. Sort the results in ascending order and provide screenshots of the query you used as well as one of the resulting data.</p>	2																									
6	<p>Using the SIS database, write a query to retrieve the following from the IncidentalFees table:</p> <ul style="list-style-type: none"> - Item (English column) - Amount/Semester - Total amount if you paid the amount for 3 semesters in a row <p>Include an alias on the item column (name it “incidentalFeeItem” and the calculated “totalAmount” column. Sort the results in ascending order by item and provide screenshots of both the query used and the resulting data.</p>	2																									
	Assignment Format	1																									
	Total	15																									



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE®

Database Management (PROG8080)

Fall 2019 - Week 2 – Monday Lesson



Agenda

- Daily Check-In / Administrivia
- Terminology Review
- Relational Database Types
- Database Configuration
 - SSMS Configuration/Exploration
- Various Database Objects
- Database Data Types
- Summary



Daily Check-In

- MS SQL Server and SSMS Installations
- eConestoga Use
- Use www.menti.com to ask me anything
 - Code is on the board
- Weekly Assignment is due on Thursday
 - New Assignment Cover Page



Fun “Data” of the Day

Thomas Alva Edison patented almost 1,300 inventions in his lifetime.

- The human eye blinks an average of 4,200,000 times a year.



Terminology To-Date

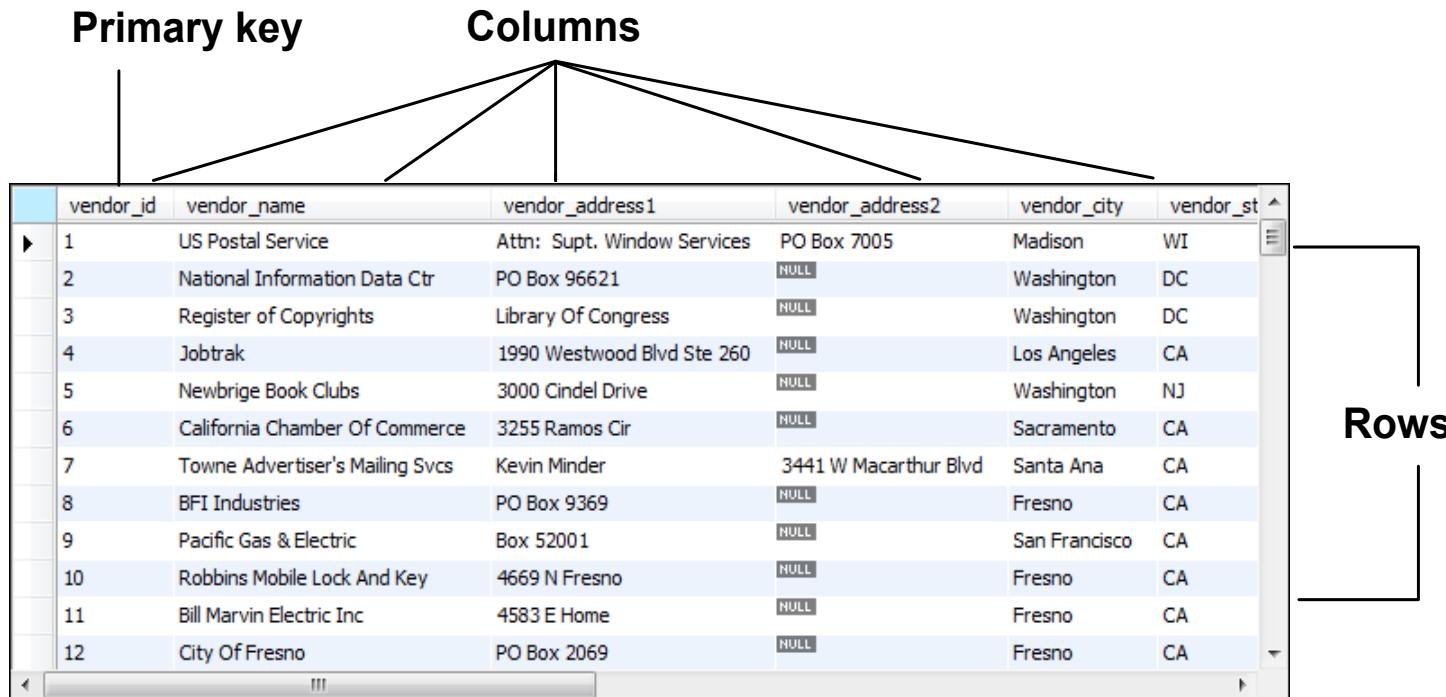
- Data
- Database (DB)
- Database Management System (DBMS)
- Relational DBMS (RDBMS)
- SQL Server
- SSMS
- SQL
- Transact-SQL (T-SQL)





Review

- Data in RDBMS Organized as Tables





New Terms

- Field
 - A single piece of information in a table
- Record
 - One complete set of *fields* in a table (row)
- File
 - Collection of *records* that make up the table

System Administrator Account

- “sa” is the default SQL Server administration account
 - Has full “rights” to do database operations
 - By default it is hidden in SQL Server
 - We need to change that in our installation
 - See the “Week 2 – Enabling SQLServer sa Account” document in eConestoga

Relational Database Types

- Online Transaction Processing (OLTP)
 - Databases built for operational requests
 - Characteristics
 - Process results quickly with minimal delay
 - Handles day-to-day types of requests
 - Process large volumes & types of transactions
 - Maintains data *integrity*
 - Original data source
 - Data is “current”



Relational Database Types - II

- Online Analytical Processing (OLAP)
 - Databases built for analytical requests
 - Characteristics
 - Handles analytical requests for decision support, reporting (read only)
 - Handles large volumes of data
 - Many data sources (OLTP and External)
 - Data is “historical”



Other New Terminology

- Data Warehouse (DW)
 - Another relational database
 - Structured to fit the need of specific requests
 - Usually a collection of data extracted, transformed and loaded (ETL) from operational databases
 - Build for analysis (an OLAP Database)



Database Set-Up

- We'll use two “fictional” databases during the course
 - SIS (Student Information System)
 - AdventureWorks
 - Microsoft provided sample database

SIS Database Set-Up

- Copy the CreateSISDatabase.sql file from eConestoga to your laptop
- Open SSMS and connect to your database **as the “sa” account**
- From the “File” menu, “Open” a “File” and browse to the CreateSISDatabase.sql file
- Take a moment to look at the file
- Use the “Execute” button (or F5) to run it



SIS Database

- Note a few things in SSMS about SQL file
 - Colour-coded statements for easy reference
 - Command (PRINT) to log information
 - Sequence of commands followed by “GO”
 - Use of conditions and various functions
 - By the end of this course this will all make sense!



AdventureWorks

- Larger database so unwieldy to set up using a SQL script
 - Uses a “backup” of the database instead
 - Go to the Microsoft download site and download the OLTP download for AdventureWorks2017 (a .bak file):

AdventureWorks Download

AdventureWorks Set-Up

- Open SSMS and connect as “sa”
- Install directions (from download site):
 - Right-click on “Databases” item and select “Restore Database”
 - Select “Device” and click the ellipses (...)
 - In the “Select Backup Devices” dialog, click “Add”, find the .bak file you downloaded and click Ok
 - Note: Data and log file locations should be fine

Explore the Databases

- Spend a few minutes playing with Object Explorer window in SSMS
 - Expand list of tables
 - Expand table to show properties
 - Right-click on table to get options to perform operations on the table
 - Find the SQL Server logs (In “Management”)
 - Play with the Object Explorer



Try This!

- I. Try opening a new query window (what's the short-cut?)
- II. Try changing the “active” database
- III. Configure SSMS to open up with the Object Explorer (left pane) and Query Window (Note other preferences)
- IV. More Advanced: See if you can find information on disk usage by table for the AdventureWorks database



Database Organization

- Database is made up of various “Objects”
 - Table – collection of related data
 - View – set of data based on a stored query
 - Synonym – alias or alternate name for object
 - Stored Procedure – subroutine available to database users
 - Function – subroutine that returns a single value
 - Trigger – code automatically executed in certain conditions
 - Sequences – used to generate a number sequence

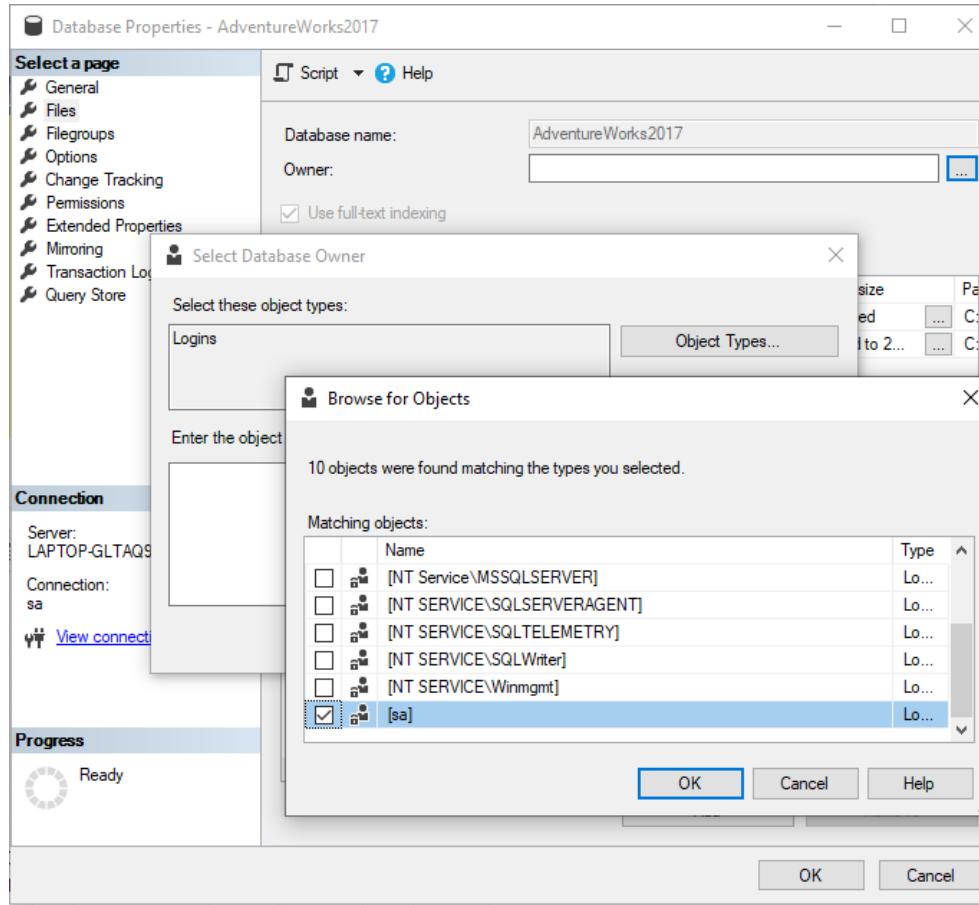


AdventureWorks Owner

- Try to create a Database Diagram for AW
 - You should get a permission warning
 - No owner specified
 - Need to access AW properties
 - Under “Files” need to set the “Owner” to “sa”



Setting AW Owner





AdventureWorks Diagrams

- Now should be able to create/save Database Diagrams for AW
 - Try creating some and see how they show relationships between tables as well as column information



Common Data Types – Part I

- Character Strings
 - Transact-SQL types:
 - char
 - Fixed/known length
 - varchar
 - Variable length
 - text
 - Large text data
 - Data not generally joined on (more on this later)

Common Data Types – Part II

- Unicode Character Strings
 - Transact-SQL types:
 - nchar, nvarchar, ntext
 - Like Character Strings but support characters for all languages



Common Data Types – Part III

- Exact Numeric
 - Exact numeric values (vary in storage and range)
 - Transact-SQL types:
 - tinyint, smallint, int, bigint
 - bit
 - Values of 0, 1 or NULL (Boolean)
 - numeric, decimal
 - smallmoney, money
 - Can use currency symbols

Common Data Types – Part IV

- Approximate Numeric
 - Numbers for which there is no exact amount
 - Scientific Notation (e.g. 2.38E-150)
 - Transact-SQL types:
 - float, real



Common Data Types – Part V

- Date and Time
 - For storing dates/times that can be translated by time zone information
 - Transact-SQL types:
 - date, datetime, datetime2, datetimeoffset, smalldatetime, time

Common Data Types – Part VI

- Binary Strings
 - For storing large string-like data (e.g. images)
 - Called BLOBS (Binary Large Objects) in some RDBMS configurations
 - Transact-SQL types:
 - binary, varbinary, image

Common Data Types – Part VII

- Other
 - Various other useful types that don't fall under the above categories
 - RDBMS-specific
 - Transact-SQL types:
 - cursor, rowversion, hierarchyid, uniqueidentifier, sql_variant, xml, Spatial Geometry Types, Spatial Geography Types, table



Common Data Types – Part VIII

- **NULL**
 - Represents data that doesn't exist in a field
 - It is not an empty string
 - It is not zero
 - It is not any value at all
 - One NULL value never equals another NULL value

Common Data Types Use

- Consult the SQL Server 2017 documentation for detailed descriptions on storage and range
 - Link in eConestoga
- Each has a purpose that best suits a situation
- **IMPORTANT** to make the right data type choice
 - Often multiple choices but usually one makes the most sense
 - Storage is **ALWAYS** a consideration



Let's Give it a Go!

- Go to www.menti.com and enter the code shown on the board
- See how you do!



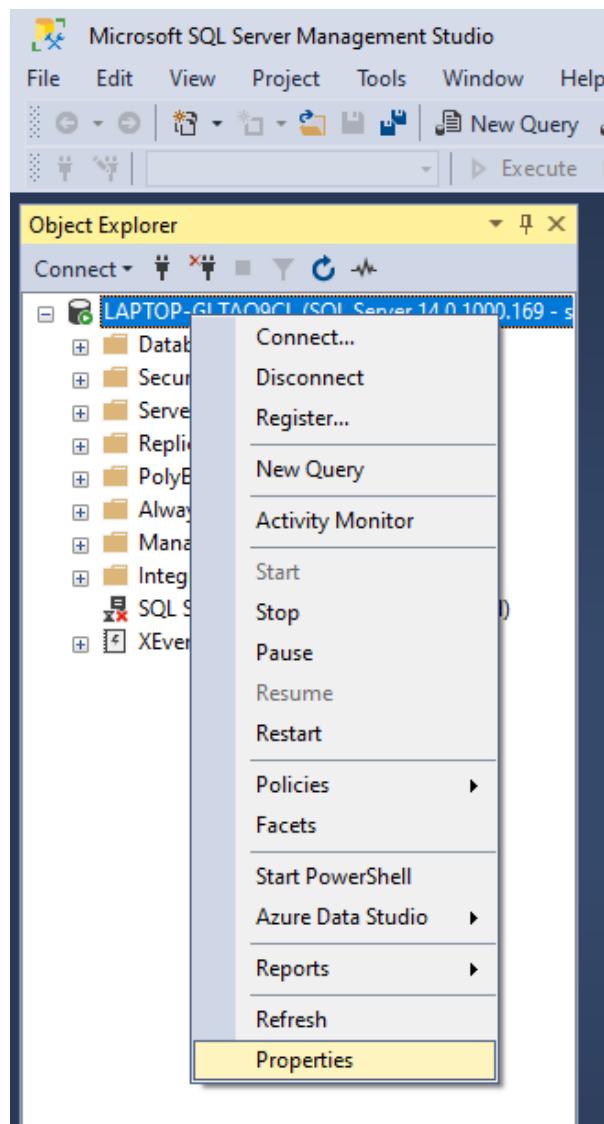
Summary

- New Terminology
- Database Objects
- Configured Databases
- Explored SSMS
- Data Types
- Assignment due Thursday
- Questions?

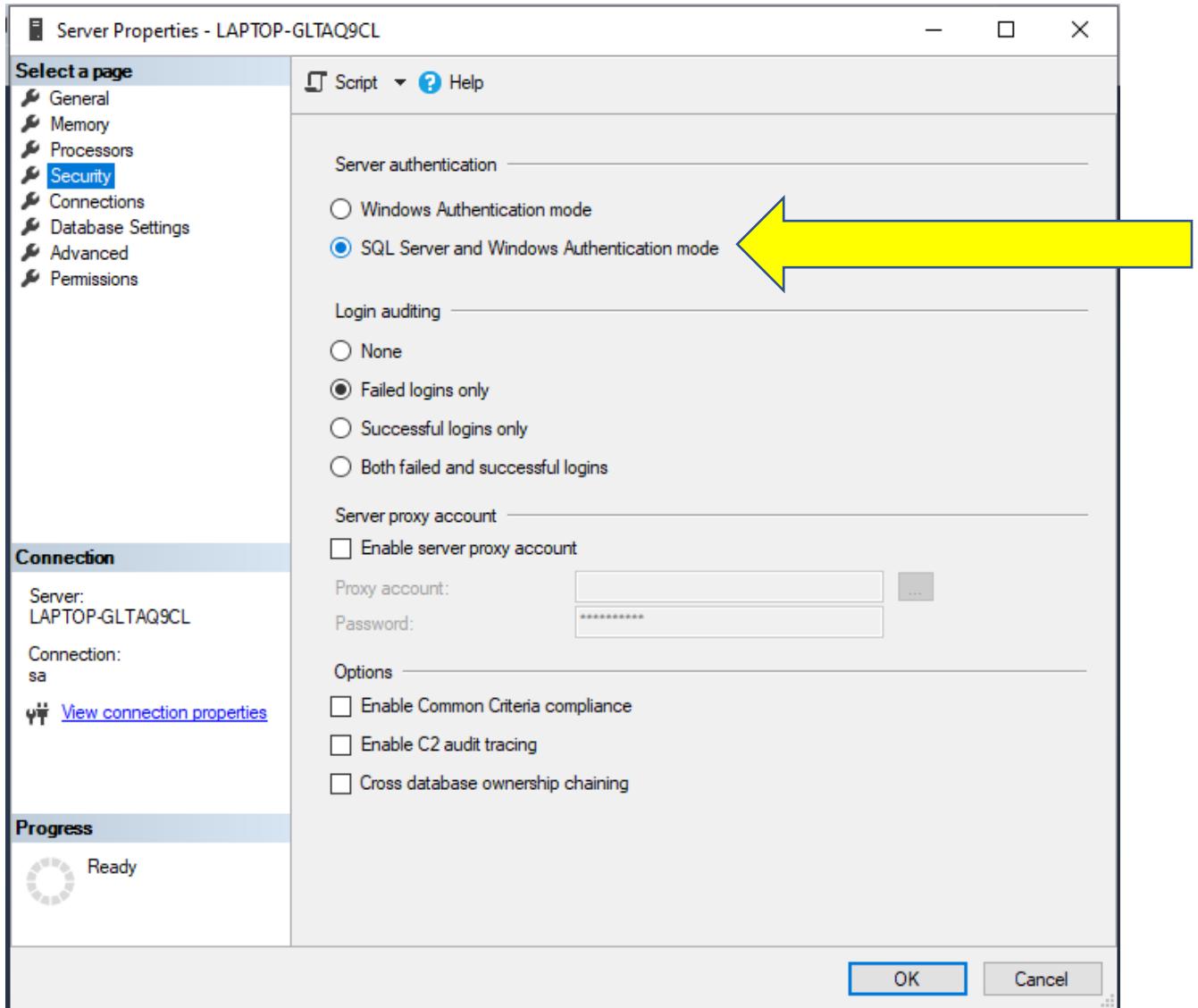
Enabling the System Administrator “sa” Account in SQL Server 2017

Steps:

1. Launch SSMS and log in.
2. Right-Click the database instance and select “Properties”.



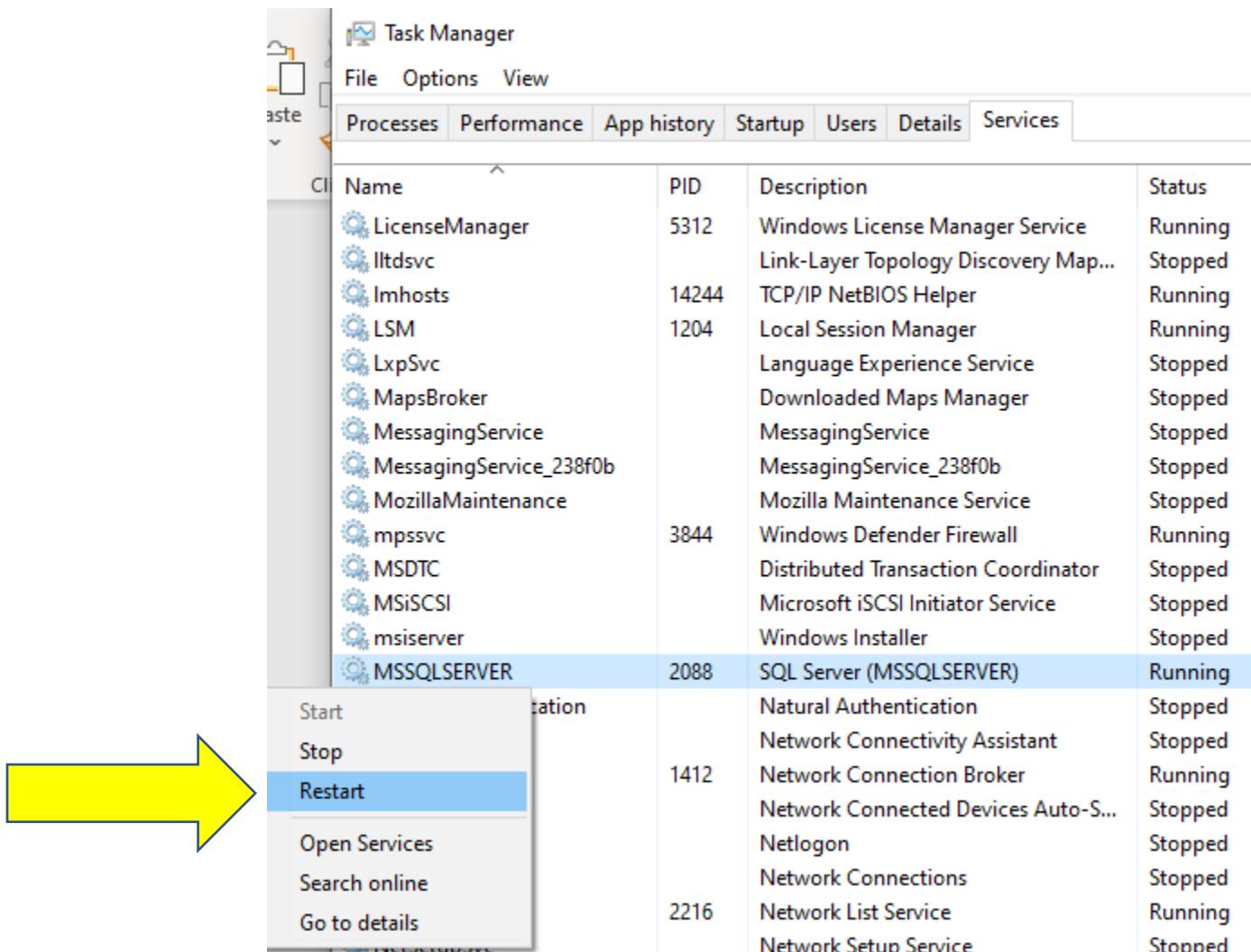
3. On the “Server Properties” page, select the “Security” option and change the “Server Authentication” selection to “SQL Server and Windows Authentication mode”. Click OK when you’re done.



4. At this point the SQL Server service needs to be restarted.
To do this:

a. Close SSMS.

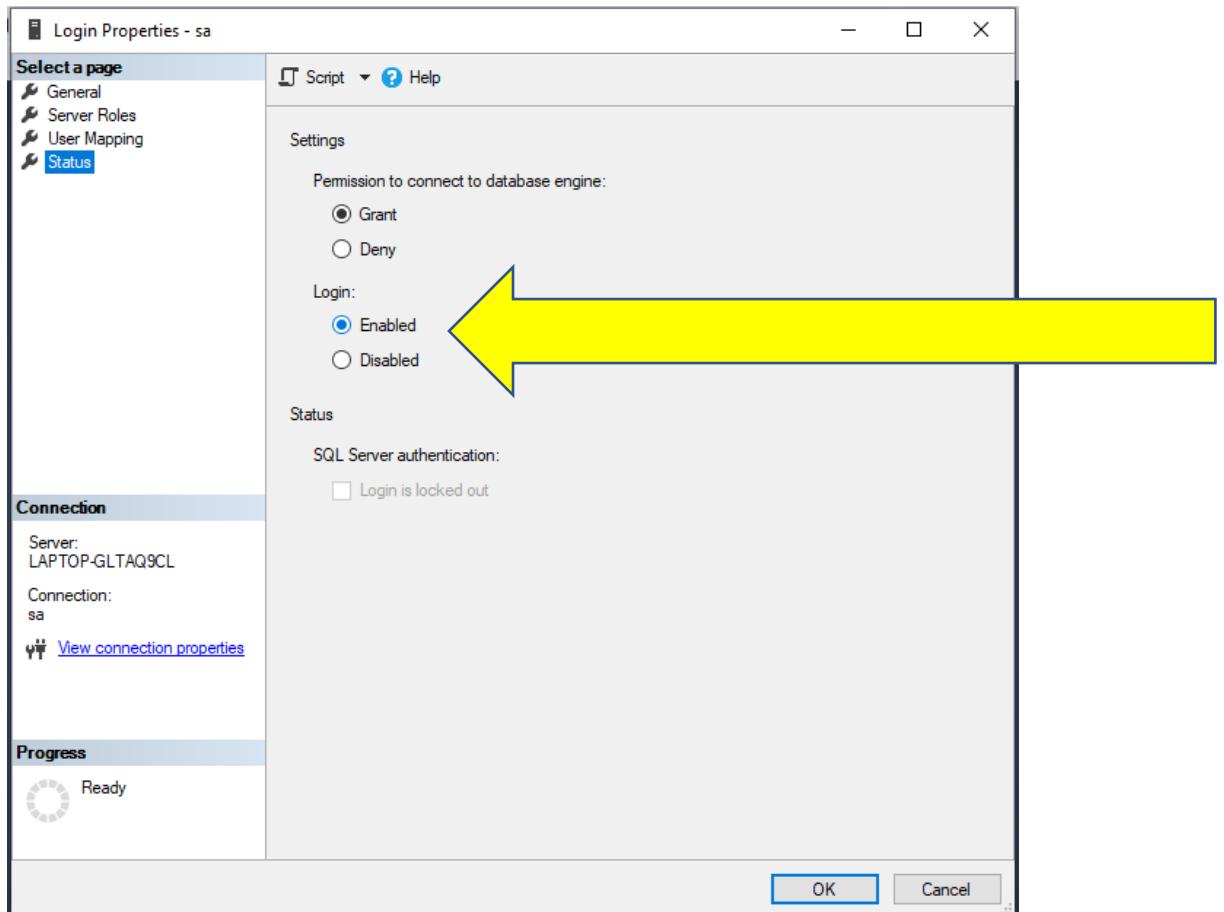
b. Go to Task Manager (Right click on the Windows icon in the task bar and select it), look under the “Services” tab and scroll down until you find the “MSSQLSERVER” service. Right-click it and choose “Restart”.



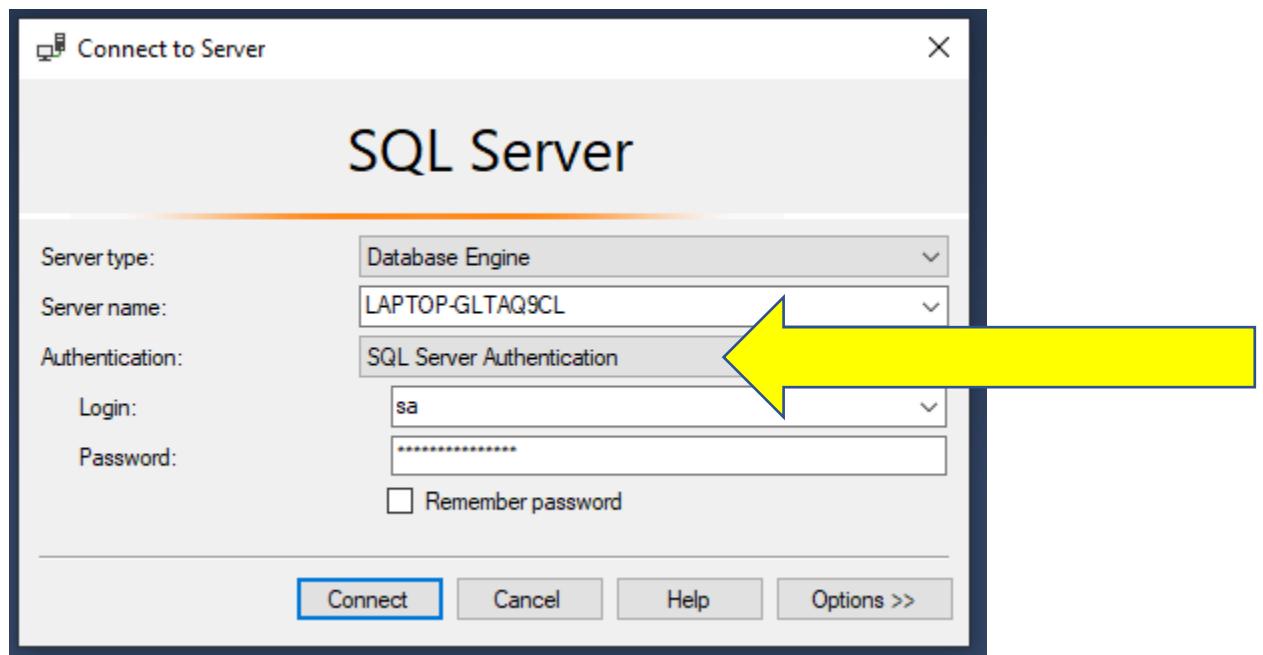
5. Re-start SSMS and log in.

6. Under the “Security” topic and “Logins” detail note that the “sa” account is disabled.

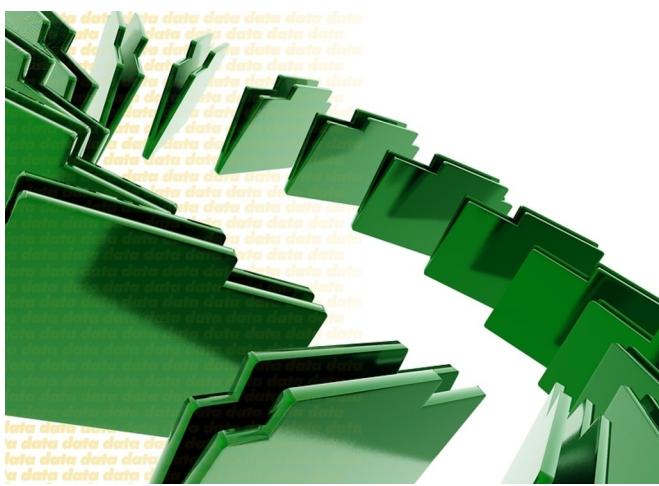
7. Right-click and go to “Properties” to get to the “Login Properties” screen
- Change the password (using a complex password for security)
 - Go to the “Status” page and change “Login” setting to “Enabled”.



8. You should now be able to connect to the database as the “sa” account. Make sure to change the “Authentication” option to “SQL Server Authentication”.



9. Celebrate!



WEEK #3

ASSIGNMENT

2% Individual Assignment

The SQL SELECT Statement

Mark Morell

Database Management – Fall 2019

Assignment Type:

- Individual – Prepare and submit your results independently

Date Due:

- Thursday, September 26th by the end of the day

Instructions:

- Please submit your assignment electronically through eConestoga.
- Assignments should be submitted as Microsoft Word files using the course coversheet format.
- If you are using external sources (images, text, etc.) you must reference them as part of your assignment and not copy them as-is.
- Best practice is to research your answers and then write the response to the question in your own words.
- Please include the question number with your responses.
- For SQL results please include print screens of the SQL used as well as the full results of the query's execution. Multiple screenshots may be required to show the full results.

Late Assignment Penalty:

Days Late	Penalty %
1	5
2	10
3	20
4	40
5	60
6	80
7	100

Assignment Questions

NOTE: Use the AdventureWorks2017 database to answer *all* of these questions

Question #	Question	Score
1	Find the number of records of people in the database who have a “Title” associated with their record.	1
2	Get a list of the <i>unique</i> last names of people in the database who have a first name of either “Ken” or “Kenneth”. Sort the last names in descending order.	2
3	Retrieve the address ID, a column with the address Line 1 and address line 2 concatenated (and formatted well) and the city for addresses that actually have a second line address and the city is either Waterloo, Toronto or Guelph. Order by city in ascending order and then by the full address in descending order.	2
4	<p>For addresses in the city of Ottawa, get the following information as separate columns:</p> <ul style="list-style-type: none"> • Address Line 1 • Address Line 2 • City • Postal Code • Date the Record was Last Modified • How many days ago the record was last modified <p>Sort the data in a way that makes sense and include a comment as to why you chose the sort order. Also be sure that the column headers all have meaningful names.</p>	2
5	We'd like to get the Last Names, First Names, Middle Name and Title for people with last names that end with "ll" (i.e. 2 L's) and a first name that has exactly four characters. Get only the first 15 records when the results are sorted by last name and then first name.	3
6	We'd like to see product inventory that we have in-stock but is low in supply and location ID of 4. Write a query to retrieve the product ID, shelf, bin and quantity where the quantity is greater than 0 but 25 or less. Sort the results as you see fit and include a comment describing why you sorted how you did.	2
7	<p>Expand upon the query in Question #6 (so keep the conditions of being at location ID 4 with a quantity greater than 0 but 25 or less) by joining your information with the Product table so that you can include the following columns in the result:</p> <ul style="list-style-type: none"> • Product Name • Product Number • Colour • Days To Manufacture • Shelf • Bin • Quantity <p>Sort these results in descending order by quantity and then by product #.</p>	3
		Total 15



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE®

Database Management (PROG8080)

Fall 2019 - Week 3 – Monday Lesson



Agenda

- Daily Check-In / Administrivia
- Terminology Review
- Many Variations of the SQL WHERE Clause
- String, Date, Data Type Manipulation
- Practice!
- Summary



Daily Check-In

- Thumb-Ball Time!
- Open Check-In With Everyone
- Current Assignment
- Use www.menti.com to ask me anything
 - Code is on the board



Fun “Data” of the Day

- Seaweed can grow up to 12 inches per day.
- A new baby usually deprives each of its parents around 350-400 hours of sleep in the first year.



Recent Terminology I

- Data Type Terms
 - Precision
 - Scale
 - Length
- Table Keys
 - Candidate Keys
 - Primary Keys
 - Natural vs. Surrogate
 - Foreign Keys
- Cardinality
- Domain
- Entity
- Data Integrity





SQL Basics Review

- Comments (-- or /* */)
- Messages (PRINT)
- Anatomy of a SQL SELECT Statement

```
SELECT [DISTINCT] {*/column/expression} [AS alias] [, ...]  
    FROM table  
    ORDER BY column [ASC/DESC] [, ...]
```

SQL WHERE Clause

- WHERE clause allows the SQL to reduce or filter the results as needed
 - e.g. I only want people out of the Person table whose first name is “Gary”
- Include the following as part of the query:

```
SELECT ...
  FROM ...
 WHERE condition
```

- Not unique to the SELECT statement



SQL WHERE Condition

- Condition is in the form:
 - *column relationalOperator value*
- *Relational Operators*
 - = *Equal To*
 - != *Not Equal To (can also use <>)*
 - > *Greater Than*
 - < *Less Than*
 - >= *Greater Than or Equal To*
 - <= *Less Than or Equal To*



SQL WHERE Example

- Question: How do I get a list of everyone in the SIS Person table whose first name is Gary?

```
USE SIS
GO
SELECT *
    FROM dbo.Person
    WHERE firstName = 'Gary'
```

- What do you notice?
 - Case sensitivity
 - Not case-sensitive by default but it CAN be set at the server, database or column level



Try This!

- See if you can write the SQL for these:
 - A. Get all of the columns from the SIS Person table who live in Kitchener

```
USE SIS
GO
SELECT *
    FROM dbo.Person
   WHERE city = 'Kitchener'
```

-- Should get 378 rows



Try This! (Part II)

- B. Get all of the columns from the SIS Person table who do not live in Canada

```
USE SIS
GO
SELECT *
    FROM dbo.Person
    WHERE countryCode != 'CAN'

-- Should get 42 rows
```



Try This! (Part III)

- C. Get the student number, transaction date and amount of Payments that are greater than \$1000

```
USE SIS
GO
SELECT studentNumber, transactionDate, amount
    FROM dbo.Payment
   WHERE amount > 1000

-- Should get 23 rows
```



Apostrophes & Quotation Marks

- Single Quotes (i.e. ‘ ’) are used to surround string values in SQL
- Generally only use Double Quotes (i.e. “ ”) when a column alias includes a space (which is generally bad practice)

Compound Expressions

- ...WHERE **NOT** *condition*
 - Means “do not include rows that meet the *condition* in the result set”
- ...WHERE *condition1* **AND** *condition2*
 - Means “include only rows that meet both *condition1* and *condition2* in the result set”
- ...WHERE *condition1* **OR** *condition2*
 - Means “include rows that meet *condition1*, *conditions2*, or both in the result set”



Compound Expression Examples

- Question: How do I get a list of everyone in the SIS Person table whose first name is NOT Gary?

```
USE SIS
GO
SELECT *
    FROM dbo.Person
   WHERE NOT firstName = 'Gary'
```



What Will These Return?

```
USE SIS
GO
SELECT *
    FROM dbo.Person
   WHERE firstName = 'Gary' AND
         city != 'Kitchener'
```

```
USE SIS
GO
SELECT *
    FROM dbo.Person
   WHERE firstName = 'Gary' OR
         city != 'Kitchener'
ORDER BY city ASC
```



Precedence

- Order of evaluation
 - Like order of operations in arithmetic:
 1. NOT
 2. AND
 3. OR
 - When in doubt, use parentheses



Handling NULL

- You cannot use = and != to compare with NULL field values
- Instead use:
 - IS NULL and IS NOT NULL

```
USE SIS
GO
SELECT *
    FROM dbo.Person
WHERE provinceCode IS NULL
```

```
USE SIS
GO
SELECT *
    FROM dbo.Person
WHERE provinceCode IS NOT NULL
```



Try This With NULL

Get the names and alternate phone numbers of people in the SIS Person table for those that have an alternate phone number.

```
USE SIS
GO
SELECT lastName, firstName, alternatePhone
    FROM dbo.Person
   WHERE alternatePhone IS NOT NULL
```

-- should get 382 rows



Alternative to OR

To look for a list of values you could use OR:

```
USE SIS
GO
SELECT firstName, lastName
    FROM dbo.Person
   WHERE firstName = 'Mark' OR
         firstName = 'Pierre' OR
         firstName = 'Shibaji'
```

...but it is easier to use IN:

```
USE SIS
GO
SELECT firstName, lastName
    FROM dbo.Person
   WHERE firstName IN ( 'Mark', 'Pierre', 'Shibaji' )
```



Alternative to \geq and \leq

To look at results between two values you could:

```
USE Adventureworks2017
GO
SELECT *
    FROM Sales.SalesorderHeader
 WHERE TotalDue >= 500 AND TotalDue <= 1000
```

...but it is easier to use BETWEEN:

```
USE Adventureworks2017
GO
SELECT *
    FROM Sales.SalesorderHeader
 WHERE TotalDue BETWEEN 500 AND 1000
```

- Can be used for all data types



LIKE, LIKE, LIKE

Used to perform a comparison between a value and column values using a wildcard operator.

```
match_expression [NOT] LIKE pattern [ESCAPE escape_character ]
```

Pattern Characters:

- Underscore (_) matches one character in the given position
- Percent (%) matches zero, one or more characters starting in the given position
- Square Brackets ([]) around characters indicate options for characters in a position
- All can be repeated as needed



LIKE Examples

What do you think these return?

```
USE ADVENTUREWORKS2017
GO
SELECT *
    FROM Sales.Store
    WHERE name like 'acc%'
```

```
USE ADVENTUREWORKS2017
GO
SELECT *
    FROM Sales.Store
    WHERE name like '%bike%'
```

```
USE ADVENTUREWORKS2017
GO
SELECT *
    FROM Sales.Store
    WHERE name like 'vi__age%'
```



More LIKEs

```
match_expression [NOT] LIKE pattern [ESCAPE escape_character ]
```

ESCAPE

- Used to identify an escape character used in the matching string
- For example, to search for any rows where the string “30%” is in the “comment” column you’d use the following in the WHERE clause:

```
WHERE comment LIKE '%30!%' ESCAPE '!'
```



String Manipulation

Various functions and operations can be used on strings

- Concatenate Strings with the + operator

```
USE SIS
GO
SELECT lastName + firstName AS fullName
FROM dbo.Person
```

- But format it so that it looks better with:

```
USE SIS
GO
SELECT lastName + ', ' + firstName AS fullName
FROM dbo.Person
```



String Functions

- There are a number of string functions available.
The most-used include:
 - LEFT(expression, length) / RIGHT(expression, length)
 - Get the left/right number of characters from a string
 - LEN(expression)
 - Get the number of characters in the string expression
 - TRIM(expression)
 - Remove space characters from both ends of a string
 - LOWER(expression) / UPPER(expression)
 - Convert the string expression to lowercase or uppercase
 - REPLACE(expression, pattern, replacement)
 - Replace occurrences of a string with another string value



Date Manipulation

- Adding/Subtracting days to/from a date

```
USE Adventureworks2017
```

```
GO
```

```
SELECT SalesorderID, orderDate, orderDate + 14 AS deliveryDate
FROM Sales.SalesOrderHeader
```

- Date Functions
 - GETDATE()
 - Gets the current date/time (from the server)

```
SELECT GETDATE()
```



Date Functions

- Date Functions
 - DATEPART(*datepart*, date)
 - Gets a part of the date from a date (as an int)
 - Common *datepart* values are
 - » *year*
 - » *month*
 - » *day*
 - » *dayofyear*
 - » *week*
 - » *hour*
 - » *minute*

```
SELECT DATEPART( year, GETDATE() )
```



Date Functions II

- DATEADD(*datepart*, amount, date)
 - Adds an amount of the *datepart* to the date
 - Amount can be negative (to subtract from date)

```
SELECT DATEADD( year, 1, GETDATE() ) AS oneYearAway
```

- DATEDIFF(*datepart*, date1, date2)
 - Returns the difference between two dates based on the datepart unit specified

```
SELECT DATEDIFF( month, '2017/02/25', GETDATE() ) AS DateDiff
```



Data Type Conversion

- Convert an expression of one data type to another
 - `CAST (expression AS data_type [(length)])`
 - `CONVERT(data_type [(length)],
expression [, style])`

```
USE Adventureworks2017
```

```
GO
```

```
SELECT 'Sales Order #' + CONVERT( char(5), salesorderID, 1)
      + ', dated ' + CONVERT( char(8), orderDate, 1)
      + ' for $' + CONVERT( varchar(15), TotalDue, 1 )
      AS salesSummary
  FROM Sales.SalesOrderHeader
```



Other Handy SQL Concepts

- Limiting Result-Set
 - Want the top x records? Use:

```
USE SIS
GO
SELECT TOP 5 lastName + firstName AS fullName
    FROM dbo.Person
    ORDER BY fullName
```

- Want the top x % of records? Use:

```
USE SIS
GO
SELECT TOP 5 PERCENT lastName + firstName AS fullName
    FROM dbo.Person
    ORDER BY fullName
```



Other Handy SQL Concepts II

- Limiting Result-Set (continued)
 - Want the top x records but include ties? Use:

```
USE SIS
GO
SELECT TOP 56 WITH TIES firstName
    FROM dbo.Person
    ORDER BY firstName
```

- Want a COUNT of the number of records?

```
USE SIS
GO
SELECT COUNT(*) AS totalRecords
    FROM dbo.Person
```



Other Handy SQL Concepts III

- Retrieving a range of rows

```
ORDER BY order_by_list
OFFSET offset_row_count {ROW|ROWS}
[FETCH {FIRST|NEXT} fetch_row_count {ROW|ROWS} ONLY]
```

- So if you just wanted the first 15 rows:

```
USE SIS
GO
SELECT lastName, firstName
    FROM dbo.Person
ORDER BY lastName DESC
        OFFSET 0 ROWS      -- The # of rows to skip
        FETCH FIRST 15 ROWS ONLY
```

Try Writing SQL For These

- A. How many countries are identified in the SIS DB?
- B. Get the first and last names (combined in one field with lastname, firstname) of everyone in the SIS Person table that have the letters “AN” in their first name. Sort by last name
- C. Show the current date and calculate how many days it is until your next birthday
- D. In the SIS DB retrieve the list of US States that are alphabetically before the letter “M”. Order by name
- E. Get rows 20-35 in the SIS Person table where the last name has “ar” in it. Show the last name and then the last name where the “ar” are replaced with “ehr”



Summary

- Questions?
- Assignment Reminders



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

Fall 2019 - Week 3 – Thursday Lesson



Agenda

- Daily Check-In / Administrivia
- Terminology Review
- Last Week's Questions Review
- SQL Coding Standards
- Basic Table Joins
- Practice!
- Summary



Daily Check-In

- Thumb-Ball Time!
- Open Check-In With Everyone
- Current Assignment Status
- Use www.menti.com to ask me anything
 - Code is on the board



Fun “Data” of the Day

Source: <https://www.livin3.com/50-cool-and-weird-fun-facts-that-you-should-know>

The average person spends 6 months of their lifetime waiting on a red light to turn green.

One in every five adults believe that aliens are hiding in our planet disguised as humans..

Recent Terminology & Topics

- SQL WHERE clause
 - Relational Operators (=, !=, etc.)
 - Compound Expressions (NOT, AND, OR)
 - Handling our friend NULL
 - Using IN, BETWEEN and LIKE
 - String Manipulation and Functions
 - Date Functions
 - Data Type Conversion
 - Handy SQL (TOP, COUNT and OFFSET)





SQL Review

Anatomy of a SQL SELECT Statement:

```
SELECT [DISTINCT] {*/column/expression} [AS alias] [,...]
    FROM table
    WHERE condition
    ORDER BY column [ASC/DESC] [,...]
        {OFFSET offsetNumRows {ROW|ROWS}
        {FETCH {FIRST|NEXT} fetchNumRows {ROW|ROWS} ONLY}
```



Last Week's Questions Review - I

A. How many countries are identified in the SIS DB?

```
USE SIS
GO
SELECT COUNT( DISTINCT name ) AS numCountries
FROM dbo.Country
```



Last Week's Questions Review - II

- B. Get the first and last names (combined in one field with last name, first name) of everyone in the SIS Person table that have the letters “AN” in their first name. Sort by last name

```
USE SIS
GO
SELECT lastName + ', ' + firstName AS fullName
    FROM dbo.Person
   WHERE firstName like '%AN%'
ORDER BY lastName
```



Last Week's Questions Review - III

C. Show the current date and calculate how many days it is until your next birthday

```
USE SIS
GO
SELECT CONVERT( date, GETDATE() ) AS currentDate,
       DATEDIFF( day, GETDATE(), '02/25/2020' ) AS daysUntilBirthday
```



Last Week's Questions Review - IV

D. In the SIS DB retrieve the list of US States that are alphabetically before the letter “M”. Order by name

```
USE SIS
GO
SELECT name AS stateName
    FROM dbo.Province
   WHERE countryCode = 'USA' AND
         name < 'M'
ORDER BY stateName
```



Last Week's Questions Review - V

- E. Get rows 20-35 in the SIS Person table where the last name as “ar” in it. Show the last name and then the last name where the “ar” are replaced with “ehr”

```
USE SIS
GO
SELECT lastName, REPLACE( lastName, 'ar', 'ehr' ) AS newLastName
  FROM dbo.Person
 WHERE lastName LIKE '%ar%'
ORDER BY lastName
    OFFSET 20 ROWS
    FETCH NEXT 15 ROWS ONLY
```



SQL Coding Standards

DO

- Use meaningful names and identifiers (singular nouns)
- Make good use of white space and indentation
- Keep code as simple as possible
- Include comments
- Use “camelCase” identifiers
- Use uppercase SQL keywords and functions

TRY TO AVOID

- Plurals for identifiers. Use the collective terms instead
- Writing the entire statement on a single line
- Using brackets unnecessarily
- Quoted identifiers (required when using a space in the identifier)

SQL Coding Standards Guide

- See *SQLCodingStandards.docx* document in eConestoga
 - Identifies 8 grading items (SQL1 through SQL8)
 - Use as an assignment and testing reference



A Note About SQL Standardization

- SQL Originated in the 1970s
- History of Standardization
 - SQL Standardized originally in ANSI X3.135 in 1986
 - Adopted by ISO (International Organization for Standardization) a few months later as ISO 9075-1987
 - Current International Standard is ISO/IEC 9075-2016
 - IEC is the International Electrotechnical Commission

Vendor Compliance to Standards

- Generally the different RDBMS vendors follow the standard
- At the SQL level differences do exist, most notably with function names and how date-related data types are used
- DB2 is arguably the closest to the ISO SQL Standard



Basic Joins

- Want data from more than one table
 - e.g. How do I get a list of all courses and when they are offered?

Course	
number	
hours	
credits	
name	
frenchName	

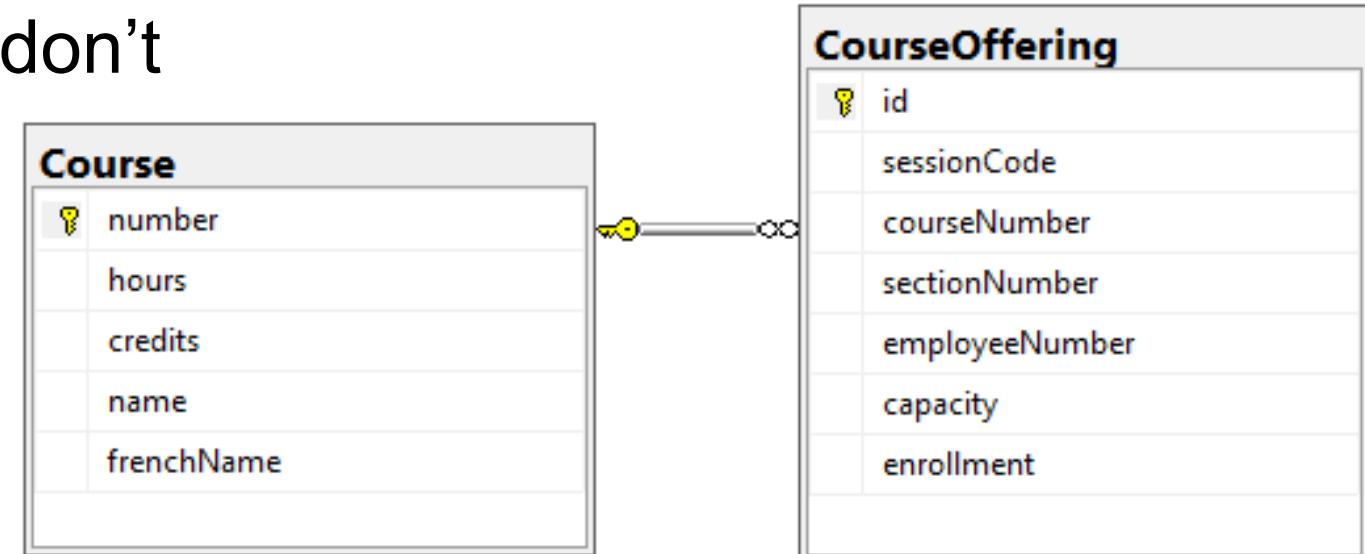
CourseOffering	
id	
sessionCode	
courseNumber	
sectionNumber	
employeeNumber	
capacity	
enrollment	



Basic Joins II

Tables have a relation to one another:

- Course.number matches with CourseOffering.courseNumber
- Note the column names don't necessarily match
- “Master” / “Parent” table
- “Detail” / “Child” table





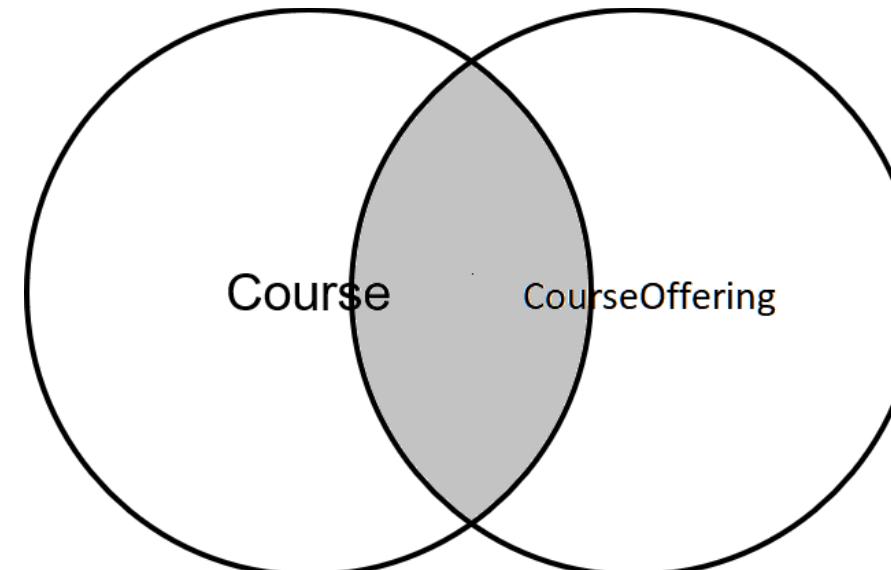
Basic Join III

- How do we get data from both tables?
 - ANSWER: JOIN them in a SELECT query
 - Many Types of Joins
 - Equijoins
 - Cross Joins
 - Reflexive Joins
 - Semi/Anti Joins



Equijoins – Inner Join

- Uses the equality operator
- “Inner Join”
 - Gets the intersection of two tables

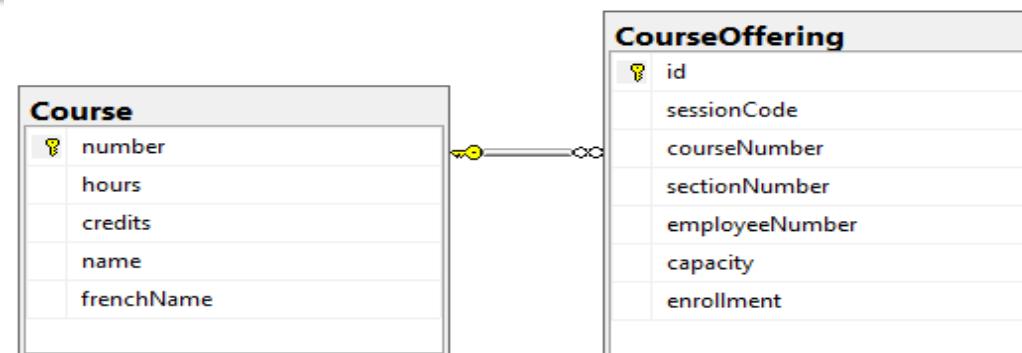
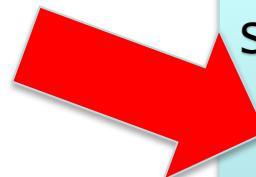




Equijoins – Inner Join Syntax

- Two syntax styles
 - “Standard” Style

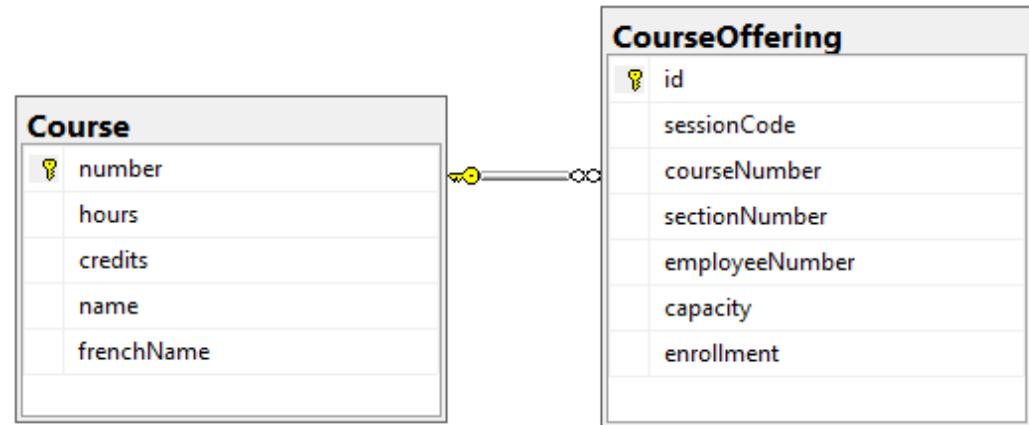
```
USE SIS
GO
SELECT Course.name, Courseoffering.sessionCode, Courseoffering.employeeNumber
    FROM dbo.Course
        INNER JOIN CourseOffering ON Course.number = Courseoffering.courseNumber
            ORDER BY Course.name, CourseOffering.sessionCode
```





Equijoins – Inner Join Syntax II

- Two syntax styles
 - “Old-School” Style



```
USE SIS
GO
SELECT Course.name, CourseOffering.sessionCode, CourseOffering.employeeNumber
  FROM dbo.Course, dbo.CourseOffering
 WHERE Course.number = CourseOffering.courseNumber
ORDER BY Course.name, CourseOffering.sessionCode
```





Equijoins – Inner Join Notes

- Be familiar with both styles
- Use aliases to make queries “simpler”

```
SELECT c.name, co.sessionCode, co.employeeNumber
  FROM dbo.Course AS c
 INNER JOIN CourseOffering AS co ON c.number = co.courseNumber
 ORDER BY c.name, co.sessionCode
```

```
SELECT c.name, co.sessionCode, co.employeeNumber
  FROM dbo.Course c, dbo.CourseOffering co
 WHERE c.number = co.courseNumber
 ORDER BY c.name, co.sessionCode
```

Equijoins – Inner Join Notes II

- Ambiguous Columns
 - If the same column name appears in more than one table in a join and you don't specify the table name it is *ambiguous*
 - To be sure always prefix all columns with the table name
- Complex Joins
 - You can join as many tables as you like
 - Be aware of the potential for degraded performance the more tables included



Equijoins – Joins and WHERE Clause

- The WHERE clause can also include other conditions to limit the results even further

```
SELECT c.name, co.sessionCode, co.employeeNumber
  FROM dbo.Course AS c
 INNER JOIN CourseOffering AS co ON c.number = co.courseNumber
 WHERE co.sessionCode like 'F%'      -- To limit results to those in the Fall
 ORDER BY c.name, co.sessionCode
```

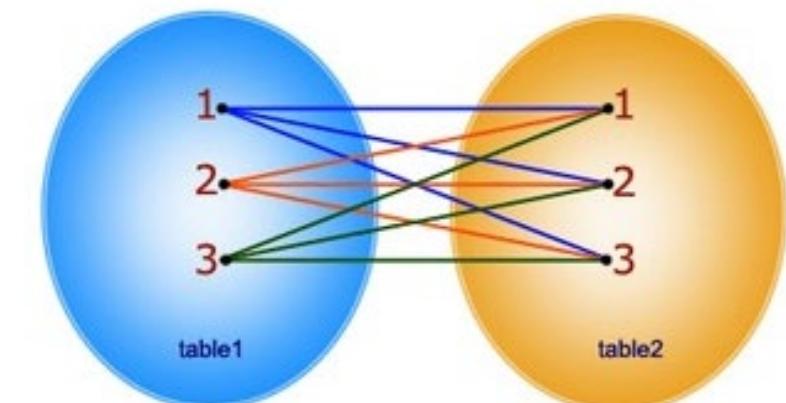
```
SELECT c.name, co.sessionCode, co.employeeNumber
  FROM dbo.Course c, dbo.Courseoffering co
 WHERE c.number = co.courseNumber
   AND co.sessionCode like 'F%' -- To limit results to those in the Fall
 ORDER BY c.name, co.sessionCode
```



Cross Join (Cartesian Product)

- If you don't provide a join condition, *every* row in the first table will be joined with *every* row in the second table
 - Generally not useful but there are rare occasions to use it
 - Can you think of one?

```
SELECT c.name, co.sessionCode, co.employeeNumber
  FROM dbo.Course c, dbo.Courseoffering co
 ORDER BY c.name, co.sessionCode
```





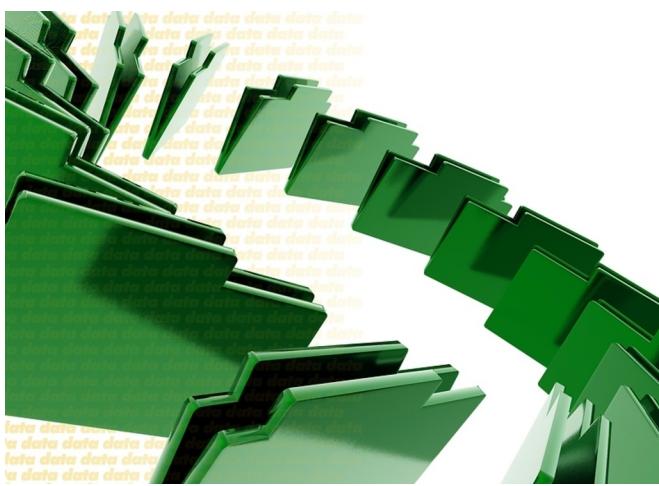
Practice Time!

- A. Get the employee number and full name (last name, first name) of every employee in the SIS database
- B. In SIS, get a distinct list of the full name of countries that people are from in the Person table (sort by country)
- C. In SIS get the full names of every person who is a student who lives on a street with a “1” in the address
- D. In SIS list the full names of software products used in a lab with a roomId of 6.
- E. Using the SIS database, retrieve the list of courses that are offered in the “F09” session and list the course name and the name of the employee teaching it. (*Tricky!*)



Summary

- Review Session Concepts
- This Week's Assignment
- Current Assignment Reminder
- Questions?



WEEK #4

ASSIGNMENT

2% Individual Assignment

Work the JOIN!

Mark Morell

Database Management – Fall 2019

Assignment Type:

- Individual – Prepare and submit your results independently

Date Due:

- Thursday, October 3rd by the end of the day

Instructions:

- Please submit your assignment electronically through eConestoga.
- Assignments should be submitted as Microsoft Word files using the course coversheet format.
- If you are using external sources (images, text, etc.) you must reference them as part of your assignment and not copy them as-is.
- Best practice is to research your answers and then write the response to the question in your own words.
- Please include the question number with your responses.
- For SQL results please include print screens of the SQL used as well as the full results of the query's execution. Multiple screenshots may be required to show the full results.

Late Assignment Penalty:

Days Late	Penalty %
1	5
2	10
3	20
4	40
5	60
6	80
7	100

Assignment Questions

NOTE: Use the AdventureWorks2017 database to answer *all* of these questions

Question #	Question	Score
1	<p>We want to get information on all <u>employees</u> that Human Resources keeps records on. For every employee who has a marital status of <i>single</i> and has “WA” in their last name, get the following information:</p> <ul style="list-style-type: none"> • The person’s ID (<i>BusinessEntityID</i> in the tables) • The person’s full name as one field • The person’s job title • The person’s email address (join to another table) <p>(Hint: This query will result in joins with 3 tables. You do <i>not</i> need to use an outer join)</p>	3
2	<p>We want to look at the Production schema data (so the table names are all prefaced with “Production.”) to get information on all Products that are white in colour. Show the:</p> <ul style="list-style-type: none"> • Product ID • Product Name • List Price <p>Sort the data by Product ID.</p>	2
3	<p>Now we want to expand upon the above result to show reviews for those products that have them (Hint: Use the <i>ProductReview</i> table). Show all of the products that you found above but include the following information if they exist for each:</p> <ul style="list-style-type: none"> • Product Review Rating • Product Review Comments <p>Sort the data by Product ID. (Hint: this will use an outer join)</p>	3
4	Find the maximum freight amount paid from the <i>SalesOrderHeader</i> table (in the Sales schema)	2
5	From the <i>SalesOrderHeader</i> table, write a single query to determine the number of sales (number of records) and the total (sum) of sales (<i>TotalDue</i>) for Customer ID 30117	2
6	Also from the <i>SalesOrderHeader</i> table, write a query to show the sum of the <i>TotalDue</i> by <u>CustomerID</u> having a sum greater than \$750,000. Sort the result by the <i>TotalDue</i> in descending order. Your results will list each Customer ID in the database along with the sum of their sales.	3
	Total	15



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

Fall 2019 - Week 4 – Monday Lesson



Agenda

- Daily Check-In / Administrivia
- Terminology/Topic Review
- Last Week's Questions Review
- Self/Reflective Joins
- Outer Joins
- Practice!
- Summary



The Daily Dose

- Thumb-Ball Time!
- Check-In With Everyone
- Current Assignment Status
- Use www.menti.com to ask me anything
 - Code is on the board



Fun “Data” of the Day

Source: <https://www.livin3.com/50-cool-and-weird-fun-facts-that-you-should-know>

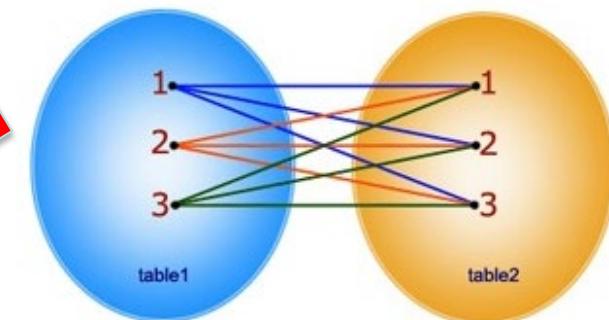
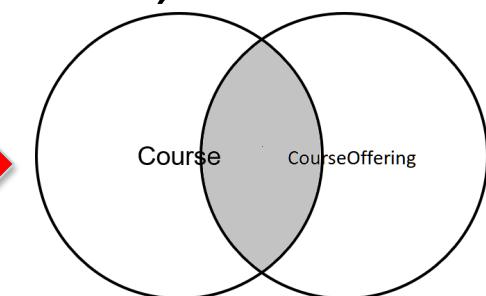
You are 1% shorter in the evening than in the morning

To produce a single pound of honey, a single bee would have to visit 2 million flowers



Recent Terminology & Topics

- SQL Coding Standards
 - Meaningful, well-commented and spaced code
- SQL Standardization (ISO/IEC 9075-2016)
- Basic Joins
 - Equijoins – Inner Join
 - Cross Join
- Syntax Styles
- Ambiguous Column Names





SQL Review

Anatomy of an Inner Join SELECT Statement (both styles):

```
SELECT [DISTINCT] {*/table1.column/table2.column/expression} [AS alias] [,...]
  FROM table1
 INNER JOIN table2 ON table1.column = table2.column
 WHERE conditions
 ORDER BY column [ASC/DESC] [,...]
 {OFFSET offsetNumRows {ROW|ROWS}
 {FETCH {FIRST|NEXT} fetchNumRows {ROW|ROWS} ONLY}
```

```
SELECT [DISTINCT] {*/table1.column/table2.column/expression} [AS alias] [,...]
  FROM table1, table2
 WHERE table1.column = table2.column AND
       conditions
 ORDER BY column [ASC/DESC] [,...]
 {OFFSET offsetNumRows {ROW|ROWS}
 {FETCH {FIRST|NEXT} fetchNumRows {ROW|ROWS} ONLY}
```



Last Week's Questions Review - I

- A. Get the employee number and full name (last name, first name) of every employee in the SIS database

```
SELECT e.number AS employeeNumber, p.lastName + ', ' + p.firstName AS employeeName
  FROM dbo.Employee AS e
 INNER JOIN dbo.Person AS p ON p.number = e.number
 ORDER BY employeeName ASC
```

```
SELECT e.number AS employeeNumber, p.lastName + ', ' + p.firstName AS employeeName
  FROM dbo.Employee AS e, dbo.Person as p
 WHERE p.number = e.number
 ORDER BY employeeName ASC
```



Last Week's Questions Review - II

- B. In SIS, get a distinct list of the full name of countries that people are from in the Person table (sort by country)

```
SELECT DISTINCT c.name
  FROM dbo.Country AS c
 INNER JOIN dbo.Person AS p ON p.countryCode = c.Code
 ORDER BY c.name ASC
```

```
SELECT DISTINCT c.name
  FROM dbo.Country AS c, dbo.Person AS p
 WHERE p.countryCode = c.Code
 ORDER BY c.name ASC
```



Last Week's Questions Review - III

C. In SIS get the full names of every person who is a student who lives on a street with a “1” in the address

```
SELECT p.lastName + ', ' + p.firstName AS studentName, p.street
  FROM dbo.Person AS p
INNER JOIN dbo.Student AS s ON s.number = p.number
 WHERE p.street LIKE '%1%'
ORDER BY studentName
```

```
SELECT p.lastName + ', ' + p.firstName AS studentName, p.street
  FROM dbo.Person AS p, dbo.Student AS s
 WHERE s.number = p.number
       AND p.street LIKE '%1%'
ORDER BY studentName
```



Last Week's Questions Review - IV

D. In SIS list the full names of software products used in a lab with a roomId of 6

```
SELECT s.product
  FROM dbo.Software AS s
INNER JOIN dbo.LabSoftware AS lab ON lab.softwareUniqueId = s.uniqueId
 WHERE lab.roomId = 6
ORDER BY s.product
```

```
SELECT s.product
  FROM dbo.Software AS s, dbo.LabSoftware AS lab
 WHERE lab.softwareUniqueId = s.uniqueId
   AND lab.roomId = 6
ORDER BY s.product
```



Last Week's Questions Review - V

- E. Using the SIS database, retrieve the list of courses that are offered in the “F09” session and list the course name and the name of the employee teaching it

```
SELECT c.name AS courseName, p.lastName + ', ' + p.firstName AS instructor
FROM dbo.Course AS c
INNER JOIN dbo.CourseOffering AS co ON c.number = co.courseNumber
INNER JOIN dbo.Person AS p ON co.employeeNumber = p.number
WHERE co.sessionCode = 'F09'
ORDER BY courseName ASC
```

```
SELECT c.name AS courseName, p.lastName + ', ' + p.firstName AS instructor
FROM dbo.Course AS c, dbo.CourseOffering AS co, dbo.Person AS p
WHERE c.number = co.courseNumber
    AND co.employeeNumber = p.number
    AND co.sessionCode = 'F09'
ORDER BY courseName ASC
```



Self or Reflexive Joins

- Simply an equi-join of a table to itself
 - It's like a “look-back” situation
 - Why use one?
 - Fictional Example: An employee has a supervisor identified with their record:

```
SELECT emp.name AS employeeName, manager.name AS managerName
  FROM dbo.staff AS emp
INNER JOIN dbo.staff AS manager ON manager.employeeNumber = emp.managerNumber
```

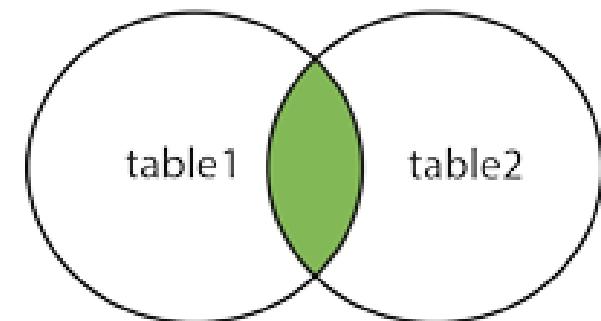


More Advanced Joins

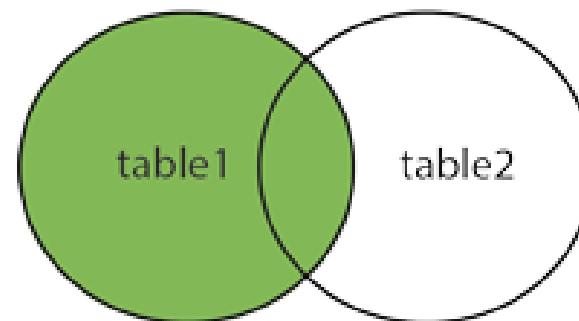
- Want data from more than one table BUT want data from one that doesn't appear in the other or maybe ALL data

Outer Joins

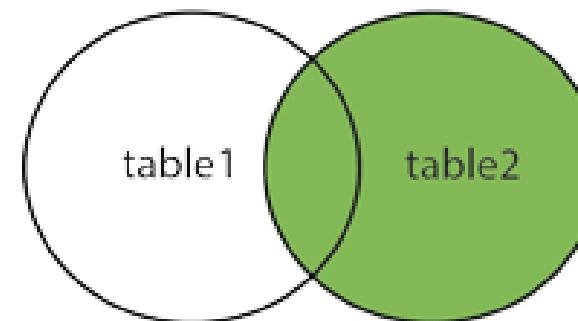
INNER JOIN



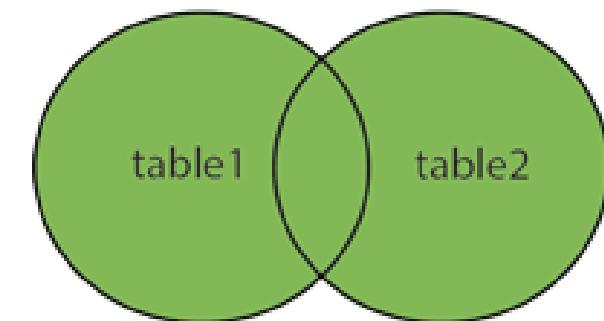
LEFT JOIN



RIGHT JOIN



FULL OUTER JOIN





Equijoins – Outer Joins

“Outer Join”

- Return *all* rows from one table and if join conditions are met, data from the other
- Resolves the issue of “lost” data from inner joins
- Three types of Outer Joins:
 - LEFT – All data from the “left” table regardless of data in the right table
 - RIGHT – All data from the “right” table regardless of data in the left table
 - FULL – All data from both tables

Outer Joins I

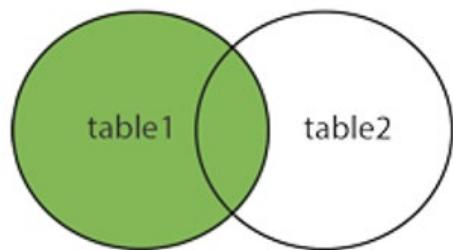
- Consider the records in the SIS Employee Table:
 - How many records are there?
 - Answer: 14
- Consider the question “What are the IDs and phone extensions of all employees and their managers?”

```
SELECT e.number, e.extension, m.number, m.extension  
      FROM dbo.employee AS e  
INNER JOIN dbo.employee AS m ON m.number = e.reportsTo
```

- How many rows did you get? Answer: 10
 - What happened to the other 4?

Outer Joins II

- The inner join removes all employees that do not have a manager record (i.e. the reportsTo is *NULL*)
- A Left Outer Join solves the problem by ensuring that half of the join is returned in its entirety



```
SELECT e.number, e.extension, m.number, m.extension
  FROM dbo.employee AS e
LEFT OUTER JOIN dbo.employee AS m ON m.number = e.reportsTo
```

- Now how many rows did you get? Answer: 14
 - What values get returned for an un-found manager entry? Answer: *NULL*



Outer Joins III

```
SELECT *
  FROM table1
LEFT OUTER JOIN table2 ON table1.x = table2.x
```

- The result here is everything from *table1* along with data that matches from *table2*
- Non-matching entries from *table2* are *NULL*
- *table1* is called the “preserved” table
- *table2* is called the “null-supplying” table
- Note: Old Style Syntax
 - Not supported in SQL Server

```
SELECT *
  FROM table1, table2
 WHERE table1.x = table2.x (+)
```



Left Outer Join – Try This!

Question: From all of the courses in the SIS database what are the pre-requisites for courses whose courseNumber begins with “COMP”? (*Hint:* Use the dbo.Course and dbo.CoursePrerequisiteAnd tables)

```
SELECT c.number, c.name, cp.prerequisiteNumber
  FROM dbo.Course AS c
LEFT OUTER JOIN dbo.CoursePrerequisiteAnd AS cp ON cp.courseNumber = c.number
 WHERE c.number like 'COMP%'
```



Outer Joins IV

– Right Outer Join

- Really provide no additional use over LEFT OUTER JOIN functionality

table1 LEFT OUTER JOIN table2

Is the same as:

table2 RIGHT OUTER JOIN table1

- Used simply for readability (and rarely used)

– Syntax

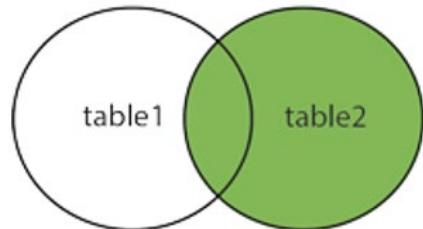
```
SELECT *
  FROM table1
RIGHT OUTER JOIN table2 ON table1.x = table2.x
```

– Returns everything from *table2* along with matching *table1* data

– Old Style Syntax

- Not supported in SQL Server

```
SELECT *
  FROM table1, table2
 WHERE table1.x (+) = table2.x
```

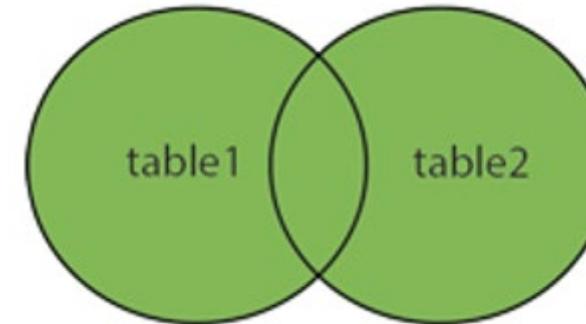




Outer Joins V

The Full Outer Join retrieves all data from both tables

- Both sides of the join are *preserved* and *null-supplied*



- Could be written as the concatenation of three results:
 - The INNER JOIN results
 - The LEFT OUTER JOIN results
 - The RIGHT OUTER JOIN results



Complete Outer Join Example – Step 1

In the SIS database, show the code and name of all schools in the dbo.School table and order it by code

```
USE SIS
GO
SELECT s.code, s.name
    FROM dbo.School AS s
ORDER BY s.code
```

	code	name
1	BUS	Business and Hospitality
2	CAA	Career and Academic Access
3	CLI	Conestoga Language Institute
4	EIT	Engineering and Information Technology
5	HLC	Health and Life Sciences and Community Services
6	LS	Liberal Studies
7	MD	Media and Design
8	TAP	Trades and Apprenticeship



Complete Outer Join Example – Step 2

Now list the schools (by code and name) and the employee ID and extension who work in those schools

```
USE SIS
GO
SELECT s.code, s.name, e.number, e.extension
    FROM dbo.School AS s
INNER JOIN dbo.Employee AS e ON e.schoolCode = s.code
ORDER BY s.code, e.number
```



Complete Outer Join Example – Step 2 Result

Now list the schools (by code and name) and the employee ID and extension who work in those schools

	code	name	number	extension
1	BUS	Business and Hospitality	6434468	9832
2	BUS	Business and Hospitality	8383259	5162
3	EIT	Engineering and Information Technology	2117745	3578
4	EIT	Engineering and Information Technology	3214350	8529
5	EIT	Engineering and Information Technology	3455730	7693
6	EIT	Engineering and Information Technology	3514239	8371
7	EIT	Engineering and Information Technology	3566163	2378
8	EIT	Engineering and Information Technology	4427732	4298
9	EIT	Engineering and Information Technology	7572373	1478
10	EIT	Engineering and Information Technology	8135170	8529
11	TAP	Trades and Apprenticeship	6860242	1795
12	TAP	Trades and Apprenticeship	6886352	6742
13	TAP	Trades and Apprenticeship	8436780	7361



Complete Outer Join Example – Step 3

Now convert the INNER JOIN to a LEFT OUTER JOIN

```
USE SIS
GO
SELECT s.code, s.name, e.number, e.extension
    FROM dbo.School AS s
    LEFT OUTER JOIN dbo.Employee AS e ON e.schoolCode = s.code
    ORDER BY s.code, e.number
```

What will these results give you?



Complete Outer Join Example – Step 3 Result

Now convert the INNER JOIN to a LEFT OUTER JOIN

What do
the *NULL*
values
mean?

	code	name	number	extension
1	BUS	Business and Hospitality	6434468	9832
2	BUS	Business and Hospitality	8383259	5162
3	CAA	Career and Academic Access	NULL	NULL
4	CLI	Conestoga Language Institute	NULL	NULL
5	EIT	Engineering and Information Technology	2117745	3578
6	EIT	Engineering and Information Technology	3214350	8529
7	EIT	Engineering and Information Technology	3455730	7693
8	EIT	Engineering and Information Technology	3514239	8371
9	EIT	Engineering and Information Technology	3566163	2378
10	EIT	Engineering and Information Technology	4427732	4298
11	EIT	Engineering and Information Technology	7572373	1478
12	EIT	Engineering and Information Technology	8135170	8529
13	HLC	Health and Life Sciences and Community Services	NULL	NULL
14	LS	Liberal Studies	NULL	NULL
15	MD	Media and Design	NULL	NULL
16	TAP	Trades and Apprenticeship	6860242	1795
17	TAP	Trades and Apprenticeship	6886352	6742
18	TAP	Trades and Apprenticeship	8436780	7361



Complete Outer Join Example – Step 4

Convert the LEFT OUTER JOIN to a RIGHT OUTER JOIN

```
USE SIS
GO
SELECT s.code, s.name, e.number, e.extension
    FROM dbo.School AS s
RIGHT OUTER JOIN dbo.Employee AS e ON e.schoolCode = s.code
ORDER BY s.code, e.number
```

What will these results give you?



Complete Outer Join Example – Step 4 Result

Convert the LEFT OUTER JOIN to a RIGHT OUTER JOIN

What do the
NULL values
mean here?

	code	name	number	extension
1	NULL	NULL	5512736	6183
2	BUS	Business and Hospitality	6434468	9832
3	BUS	Business and Hospitality	8383259	5162
4	EIT	Engineering and Information Technology	2117745	3578
5	EIT	Engineering and Information Technology	3214350	8529
6	EIT	Engineering and Information Technology	3455730	7693
7	EIT	Engineering and Information Technology	3514239	8371
8	EIT	Engineering and Information Technology	3566163	2378
9	EIT	Engineering and Information Technology	4427732	4298
10	EIT	Engineering and Information Technology	7572373	1478
11	EIT	Engineering and Information Technology	8135170	8529
12	TAP	Trades and Apprenticeship	6860242	1795
13	TAP	Trades and Apprenticeship	6886352	6742
14	TAP	Trades and Apprenticeship	8436780	7361



Complete Outer Join Example – Step 5

Finally complete the FULL OUTER JOIN

```
USE SIS
GO
SELECT s.code, s.name, e.number, e.extension
    FROM dbo.School AS s
    FULL OUTER JOIN dbo.Employee AS e ON e.schoolCode = s.code
    ORDER BY s.code, e.number
```

What will these results give you?



Complete Outer Join Example – Step 5 Result

Finally complete the FULL OUTER JOIN

What do
the *NULL*
values
mean in
this case?

	code	name	number	extension
1	NULL	NULL	5512736	6183
2	BUS	Business and Hospitality	6434468	9832
3	BUS	Business and Hospitality	8383259	5162
4	CAA	Career and Academic Access	NULL	NULL
5	CLI	Conestoga Language Institute	NULL	NULL
6	EIT	Engineering and Information Technology	2117745	3578
7	EIT	Engineering and Information Technology	3214350	8529
8	EIT	Engineering and Information Technology	3455730	7693
9	EIT	Engineering and Information Technology	3514239	8371
10	EIT	Engineering and Information Technology	3566163	2378
11	EIT	Engineering and Information Technology	4427732	4298
12	EIT	Engineering and Information Technology	7572373	1478
13	EIT	Engineering and Information Technology	8135170	8529
14	HLC	Health and Life Sciences and Community Services	NULL	NULL
15	LS	Liberal Studies	NULL	NULL
16	MD	Media and Design	NULL	NULL
17	TAP	Trades and Apprenticeship	6860242	1795
18	TAP	Trades and Apprenticeship	6886352	6742
19	TAP	Trades and Apprenticeship	8436780	7361



Practice Time!

- A. From the SIS database, use a self-join to get a list of all manager IDs along with the employee IDs of those who report to them.
- B. Expand upon the above query by including both the IDs and the names of both the manager and staff who report to them. Sort these by manager name ascending and staff descending.
- C. Using the AdventureWorks2017 database, get a list of sales customers from Territory ID 2 and Store ID 1316 and show any Sales Order IDs that they *may* have. Include customers with no sales orders. Retrieve the customer ID, sales order ID and the order date.
- D. Using AdventureWorks2017, first get a list of all Currency Codes and Names as well as the Country Region Code that they are used in. Next, be sure to include in the list all currency codes and names that aren't associated with a country region. Finally, can you get just a list of those currency codes and names that aren't associated with any region (using a LEFT OUTER JOIN)?



Summary

- Review Session Concepts
- This Week's Assignment
- Current Assignment Reminder
- Questions?



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

Fall 2019 - Week 4 – Thursday Lesson



Agenda

- Daily Check-In / Administrivia
- Terminology/Topic Review
- Another Look at Joins
- Last Week's Questions Review
- Aggregate Functions
- Practice!
- Summary



The Daily Dose

- Thumb-Ball Time!
- Check-In With Everyone
- Current Assignment Status
- Use www.menti.com to ask me anything
 - Code is on the board



Fun “Data” of the Day

Source: <https://www.livin3.com/50-cool-and-weird-fun-facts-that-you-should-know>

Earth has traveled more than 5,000 miles
in the past 5 minutes

It is physically impossible for pigs to look
up into the sky.



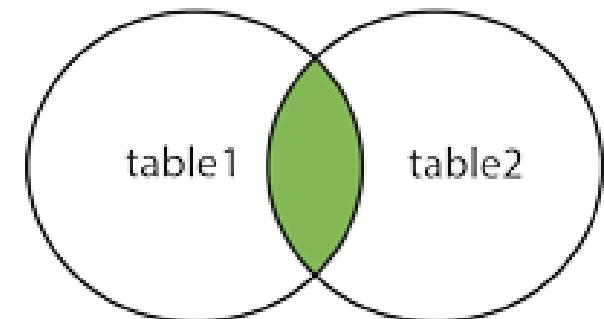
Recent Terminology & Topics

- Self / Reflexive Joins
- Various Joins
 - Inner Joins
 - Outer Joins
 - Left
 - Right
 - Full

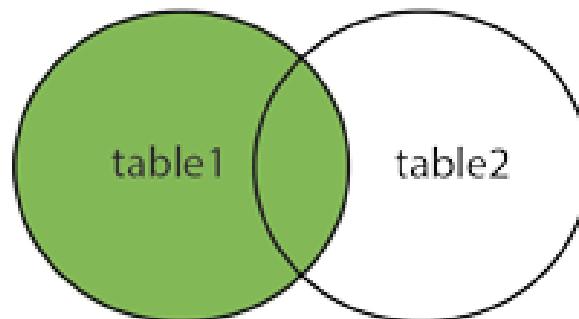


SQL Statement Anatomy Review

INNER JOIN

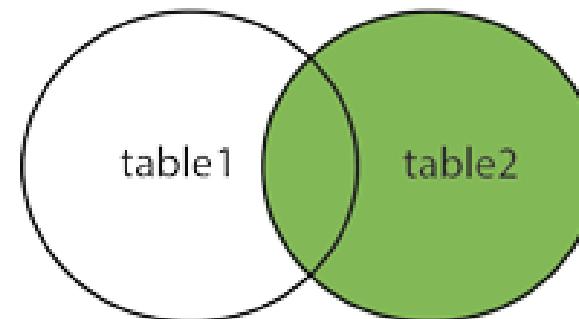


LEFT JOIN

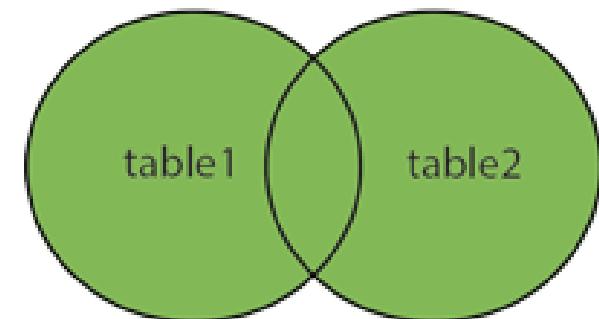


Outer Joins

RIGHT JOIN



FULL OUTER JOIN



```
SELECT [DISTINCT] {*/table1.column/table2.column/expression} [AS alias] [,...]
  FROM table1
  {INNER/LEFT/RIGHT/FULL} JOIN table2 ON table1.column = table2.column
  WHERE conditions
  ORDER BY column [ASC/DESC] [,...]
  {OFFSET offsetNumRows {ROW|ROWS}
  {FETCH {FIRST|NEXT} fetchNumRows {ROW|ROWS} ONLY}
```



Another Look at Joins

carCompany

companyId	companyName
1	Toyota
2	Ford
3	BMW
4	Honda
5	Porsche

companyMinivan

companyId	vanName
1	Sienna
2	Aerostar
2	Freestar
4	Odyssey
13	Sedona
27	Caravan

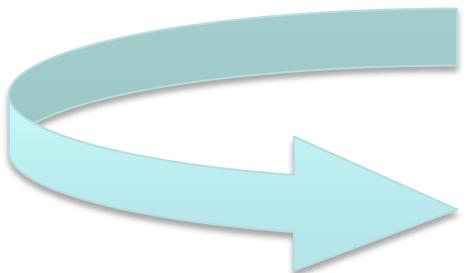


The Inner Join

- Gives the result of the intersection of both tables
 - What they have in common data-wise

carCompany	
companyId	companyName
1	Toyota
2	Ford
3	BMW
4	Honda
5	Porsche

companyMinivan	
companyId	vanName
1	Sienna
2	Aerostar
2	Freestar
4	Odyssey
13	Sedona
27	Caravan



```
SELECT cc.companyID, cc.companyName, cm.vanName
  FROM carCompany AS cc
 INNER JOIN companyMinivan AS cm ON cc.companyID = cm.companyID
```

companyId	companyName	vanName
1	Toyota	Sienna
2	Ford	Aerostar
2	Ford	Freestar
4	Honda	Odyssey

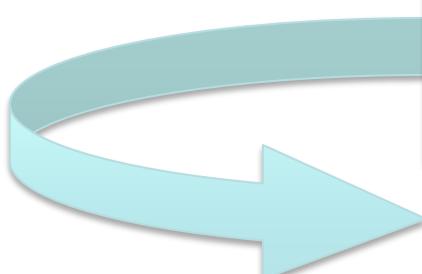


The Left Outer Join

- Gives all data from the “left” table and any matching data from the right
 - NULL values where no data exists

carCompany	
companyId	companyName
1	Toyota
2	Ford
3	BMW
4	Honda
5	Porsche

companyMinivan	
companyId	vanName
1	Sienna
2	Aerostar
2	Freestar
4	Odyssey
13	Sedona
27	Caravan



```
SELECT cc.companyID, cc.companyName, cm.vanName
  FROM carCompany AS cc
LEFT OUTER JOIN companyMinivan AS cm ON cc.companyID = cm.companyID
```

companyId	companyName	vanName
1	Toyota	Sienna
2	Ford	Aerostar
2	Ford	Freestar
3	BMW	null
4	Honda	Odyssey
5	Porsche	null

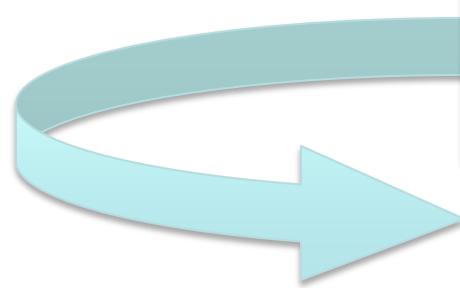


The Right Outer Join

- Gives all data from the “right” table and any matching data from the right
 - NULL values where no data exists

carCompany	
companyId	companyName
1	Toyota
2	Ford
3	BMW
4	Honda
5	Porsche

companyMinivan	
companyId	vanName
1	Sienna
2	Aerostar
2	Freestar
4	Odyssey
13	Sedona
27	Caravan



```
SELECT cc.companyID, cc.companyName, cm.vanName
  FROM carCompany AS cc
RIGHT OUTER JOIN companyMinivan AS cm ON cc.companyID = cm.companyID
```

companyID	companyName	vanName
1	Toyota	Sienna
2	Ford	Aerostar
2	Ford	Freestar
4	Honda	Odyssey
null	null	Sedona
null	null	Caravan

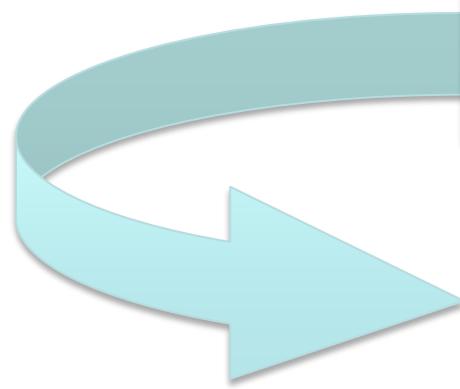


The Full Outer Join

- Gives all data from both tables and matching data where it exists. NULL otherwise

carCompany		companyMinivan	
companyId	companyName	companyId	vanName
1	Toyota	1	Sienna
2	Ford	2	Aerostar
3	BMW	2	Freestar
4	Honda	4	Odyssey
5	Porsche	13	Sedona
		27	Caravan

```
SELECT cc.companyID, cc.companyName, cm.vanName
  FROM carCompany AS cc
FULL OUTER JOIN companyMinivan AS cm ON cc.companyID = cm.companyID
```



companyId	companyName	vanName
1	Toyota	Sienna
2	Ford	Aerostar
2	Ford	Freestar
3	BMW	null
4	Honda	Odyssey
5	Porsche	null
null	null	Sedona
null	null	Caravan



Last Week's Questions Review - I

- A. From the SIS database, use a self-join to get a list of all manager IDs along with the employee IDs of those who report to them.

```
USE SIS
GO
SELECT manager.number AS managerID, staff.number AS staffID
    FROM dbo.Employee AS manager
INNER JOIN dbo.Employee AS staff ON staff.reportsTo = manager.number
ORDER BY managerID ASC, staffID ASC
```



Last Week's Questions Review - II

- B. Expand upon the above query by including both the IDs and the names of both the manager and staff who report to them. Sort these by manager name ascending and staff descending.

```
SELECT m.number AS managerID,
       mp.lastName + ', ' + mp.firstName AS managerName,
       s.number AS reporteeID,
       sp.lastName + ', ' + sp.firstName AS reporteeName
  FROM dbo.Employee AS m
 INNER JOIN dbo.Employee AS s ON s.reportsTo = m.number
 INNER JOIN dbo.Person AS mp ON mp.number = m.number
 INNER JOIN dbo.Person AS sp ON sp.number = s.number
 ORDER BY managerName ASC, reporteeName DESC
```



Last Week's Questions Review - III

- C. Using the AdventureWorks2017 database, get a list of a sales customers from Territory ID 2 and Store ID 1316 and show any Sales Order IDs that they *may* have. Include customers with no sales orders. Retrieve the customer ID, sales order ID and the order date.

```
USE Adventureworks2017
GO
SELECT c.CustomerID, s.SalesorderID, s.OrderDate
    FROM Sales.Customer AS c
LEFT OUTER JOIN Sales.SalesorderHeader AS s ON s.CustomerID = c.CustomerID
WHERE c.TerritoryID = 2
    AND c.StoreID = 1316
```



Last Week's Questions Review - IV

- D. Using AdventureWorks2017, first get a list of all Currency Codes and Names as well as the Country Region Code that they are used in. Next, be sure to include in the list all currency codes and names that aren't associated with a country region. Finally, can you get just a list of those currency codes and names that aren't associated with any region (using a LEFT OUTER JOIN)?

```
SELECT c.CurrencyCode, c.Name, crc.CountryRegionCode
  FROM Sales.Currency AS c
 INNER JOIN Sales.CountryRegionCurrency AS crc ON crc.CurrencyCode = c.CurrencyCode
```

```
SELECT c.CurrencyCode, c.Name, crc.CountryRegionCode
  FROM Sales.Currency AS c
LEFT OUTER JOIN Sales.CountryRegionCurrency AS crc ON crc.CurrencyCode = c.CurrencyCode
```

```
SELECT c.CurrencyCode, c.Name, crc.CountryRegionCode
  FROM Sales.Currency AS c
LEFT OUTER JOIN Sales.CountryRegionCurrency AS crc ON crc.CurrencyCode = c.CurrencyCode
 WHERE crc.CountryRegionCode IS NULL
```



Aggregate Functions

A function where the values of multiple rows are grouped together to form a single value

- We've already used the COUNT() function to return the # of rows
- Others include:
 - MAX() Used to determine the maximum value in a group
 - MIN() Used to determine the minimum value in a group
 - AVG() Used to determine the average of a group
 - SUM() Used to determine the sum of a group



Aggregate Functions Examples

MAX(expression):

```
-- Find the biggest sale we've had!
USE Adventureworks2017
GO
SELECT MAX( TotalDue ) AS maximumSale
FROM Sales.SalesOrderHeader
```

AVG(expression):

```
-- Find the average sale
USE Adventureworks2017
GO
SELECT AVG( TotalDue ) AS averagesale
FROM Sales.SalesOrderHeader
```

MIN(expression):

```
-- Find the oldest person
USE SIS
GO
SELECT MIN( birthdate ) AS oldest
FROM dbo.Person
```

SUM(expression):

```
-- Find the total sales in 2012
USE Adventureworks2017
GO
SELECT SUM( TotalDue ) AS totalsales
FROM Sales.SalesOrderHeader
WHERE DATEPART( year, OrderDate ) = 2012
```



Aggregate Functions Detail

Those aggregate functions work great if you're totally the entire table or data with a condition as a group but what if you want to show a group and an aggregate function of the results?

- GROUP BY and HAVING clauses



Aggregate Functions Grouping

GROUP BY:

- Allows for summarizing values
- Typically used with aggregate functions

```
-- Find the total sales for each customer
USE Adventureworks2017
GO
SELECT CustomerID, SUM( TotalDue ) AS totalsale
    FROM Sales.SalesOrderHeader
GROUP BY CustomerID
ORDER BY CustomerID
```



Aggregate Functions Group Filtering

HAVING:

- Allows you to filter GROUP BY values by the result of the aggregate function
 - Like a WHERE clause but for GROUP BY values

```
-- Find the total sales for each customer with more than $100,000
USE Adventureworks2017
GO
SELECT CustomerID, SUM( TotalDue ) AS totalSale
    FROM Sales.SalesOrderHeader
GROUP BY CustomerID
HAVING SUM( TotalDue ) > 100000
ORDER BY CustomerID
```



The UNION Operator

Allows you to combine the results of two or more SELECT statements

- UNION – Excludes duplicates from the result
- UNION ALL – Includes duplicates in the result

Useful to concatenate complicated results

```
-- Get all students and employees
USE SIS
GO
SELECT number
    FROM dbo.Employee
UNION
SELECT number
    FROM dbo.Student
```



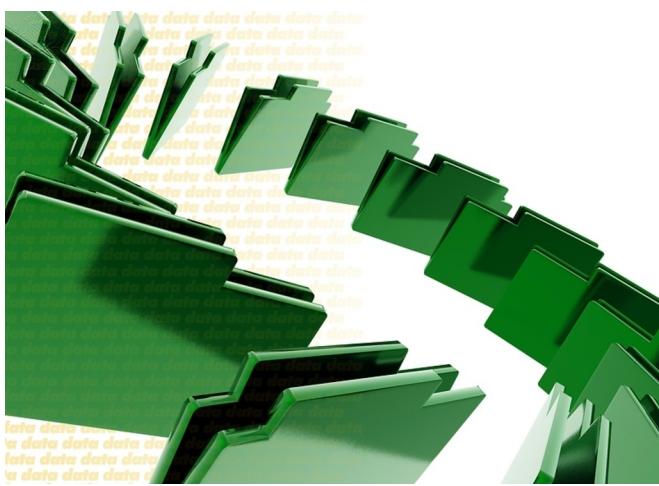
Practice Time!

- A. In AdventureWorks2017, find the Minimum Sales, Maximum Sales, Average Sales and Total Sales in the SalesOrderHeaderTable.
- B. In AdventureWorks2017, get the Sales Person ID and the total of their sales from the SalesOrderHeaderTable. Order them in descending order by total sales.
- C. Modify the above query to replace the Sales Person ID with the Sales Person first and last name (as **separate** fields) of the sales person (*Hint:* SalesPersonID is equivalent to BusinessEntityID).
- D. Modify the above query to limit the results to only sales where the order date is in 2014.
- E. Modify the above query further to limit the results to only show people who have total sales in 2014 that are greater than \$1,000,000.



Summary

- Review Session Concepts
- This Week's Assignment
- Current Assignment Reminder
- Questions?



WEEK #5

ASSIGNMENT

2% Individual Assignment

Ah, CRUD! (and subqueries)

Mark Morell

Database Management – Fall 2019

Assignment Type:

- Individual – Prepare and submit your results independently

Date Due:

- Thursday, October 10th by the end of the day

Instructions:

- Please submit your assignment electronically through eConestoga.
- Assignments should be submitted as Microsoft Word files using the course coversheet format.
- If you are using external sources (images, text, etc.) you must reference them as part of your assignment and not copy them as-is.
- Best practice is to research your answers and then write the response to the question in your own words.
- Please include the question number with your responses.
- For SQL results please include the TEXT of the SQL used as well as a print screen of your execution of the SQL and the full results of the query's execution. Multiple screenshots may be required to show everything.

Late Assignment Penalty:

Days Late	Penalty %
1	5
2	10
3	20
4	40
5	60
6	80
7	100

Assignment Questions

NOTE: Use the AdventureWorks2017 database to answer *all* of these questions

Question #	Question	Score
1	<p>Retrieve data from the HumanResources.Employee table for employees who have taken less than 8 hours of vacation (VacationHours). Show the following information in the results:</p> <ul style="list-style-type: none"> • Employee ID • Job Title • Gender converted for entries for “Male” and “Female” • The “SalariedFlag” column displayed as: <ul style="list-style-type: none"> ○ 1 = “Salaried” ○ 0 = “Non-Salaried” ○ Anything Else = “Unknown” • Vacation Hours <p>Order the data in a way that makes sense to you and include a comment to describe why you decided to sort it the way you did</p>	3
2	<p>Using a subquery in the <i>SELECT</i> portion of your query, retrieve a list of Product names from the Production.Product table that are Red in colour and get the total quantity of those products ordered from the Sales.SalesOrderDetail table. Sort your results in descending order by total quantity ordered.</p>	2
3	<p>Using a NOT IN subquery in the <i>WHERE</i> portion of your query, retrieve a list of the following data:</p> <ul style="list-style-type: none"> • Product Name • Product Number • Product Weight <p>From the Production.Product table. Get only products that are Black that do NOT have any sales in the Sales.SalesOrderDetail table. Order your results in ascending order by product name.</p>	3
4	Re-write the exact same query as #3 above using a NOT EXISTS subquery	2
5	<p>Write a SQL statement to change the phone number in the Person.PersonPhone number for the person with the ID of 305 to 555-867-5309. Also change the ModifiedDate in the record to the current date and time.</p>	2
6	<p>Write a set of SQL statements in a transaction to:</p> <ol style="list-style-type: none"> a. Delete entries in the Person.PersonPhone table where the phone number starts with the numbers 703 b. Delete entries in the Person.EmailAddress table where the email address starts with “AB” and the Email Address ID is greater than 5000 <p>Commit the transaction</p>	3
		Total 15



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

Fall 2019 - Week 5 – Monday Lesson



Agenda

- Daily Check-In / Administrivia
- Terminology/Topic Review
- Last Week's Questions Review
- Data Translation
- Subqueries
- Practice!
- Summary



The Daily Dose

- Thumb-Ball Time!
- Check-In With Everyone
- Current Assignment Status
- Use www.menti.com to ask me anything
 - Code is on the board



Fun “Data” of the Day

Source: <https://www.livin3.com/50-cool-and-weird-fun-facts-that-you-should-know>

About 150 people per year are killed by coconuts

Ketchup was used as a medicine back in the 1930's



Recent Terminology & Topics

- Outer Joins
 - Left
 - Right
 - Full
- Aggregate Functions
 - COUNT, MAX, MIN, AVG, SUM
 - GROUP BY
 - HAVING
- UNION Operator





SQL Statement Anatomy

```
SELECT [DISTINCT] {*/table1.column/table2.column/expression} {AS alias} [,...]
    FROM table1
{INNER/LEFT/RIGHT/FULL} JOIN table2 ON table2.column = table1.column
    WHERE conditions
{GROUP BY non-aggregate expression(s)}
{HAVING aggregate expression condition(s)}
    ORDER BY column [ASC/DESC] [,...]
        {OFFSET offsetNumRows {ROW|ROWS}}
        {FETCH {FIRST|NEXT} fetchNumRows {ROW|ROWS} ONLY}
```



SQL Statement Dissection

Note the order of the SQL statement:

```
SELECT ...
FROM ...
JOIN ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...
OFFSET ...
FETCH ...
```

1. SELECT the information (columns/expressions) you want
2. FROM what table(s)
3. JOIN with other table(s) you may need
4. WHERE you filter the rows that are returned
5. GROUP BY to report aggregate information in groupings
6. HAVING to further filter data based on the GROUP BY
7. ORDER BY to sort the full set of results
8. OFFSET/FETCH to only retrieve a subset of the information



Last Week's Questions Review - I

- A. In AdventureWorks2017, find the Minimum Sales, Maximum Sales, Average Sales and Total Sales in the SalesOrderHeaderTable.

```
SELECT MIN(s.TotalDue) AS minSales,  
       MAX(s.TotalDue) AS maxSales,  
       AVG(s.TotalDue) AS avgSales,  
       SUM(s.TotalDue) AS totalSales  
  FROM Sales.SalesorderHeader s
```



Last Week's Questions Review - II

- B. In AdventureWorks2017, get the Sales Person ID and the total of their sales from the SalesOrderHeaderTable. Order them in descending order by total sales.

```
SELECT s.SalesPersonID,
       SUM(s.TotalDue) AS totalsale
  FROM sales.SalesorderHeader AS s
 GROUP BY s.SalesPersonID
 ORDER BY totalsale DESC
```



Last Week's Questions Review - III

- C. Modify the above query to replace the Sales Person ID with the Sales Person first and last name as **separate** fields of the sales person (*Hint:* SalesPersonID is equivalent to BusinessEntityID).

```
SELECT p.LastName, p.FirstName,  
       SUM(s.TotalDue) AS totalsale  
    FROM sales.SalesOrderHeader AS s  
INNER JOIN Person.Person AS p ON p.BusinessEntityID = s.SalesPersonID  
   GROUP BY p.LastName, p.FirstName  
ORDER BY totalsale DESC
```



Last Week's Questions Review - IV

- D. Modify the above query to limit the results to only sales where the order date is in 2014.

```
SELECT p.LastName, p.FirstName,
       SUM(s.TotalDue) AS totalsale
    FROM sales.SalesOrderHeader AS s
INNER JOIN Person.Person AS p ON p.BusinessEntityID = s.SalesPersonID
   WHERE DATEPART( YEAR, s.OrderDate ) = 2014
GROUP BY p.LastName, p.FirstName
ORDER BY totalsale DESC
```



Last Week's Questions Review - V

- E. Modify the above query further to limit the results to only show people who have total sales in 2014 that are greater than \$1,000,000.

```
SELECT p.LastName, p.FirstName,
       SUM(s.TotalDue) AS totalsale
    FROM sales.SalesOrderHeader AS s
INNER JOIN Person.Person AS p ON p.BusinessEntityID = s.SalesPersonID
   WHERE DATEPART( YEAR, s.OrderDate ) = 2014
GROUP BY p.LastName, p.FirstName
HAVING SUM(s.TotalDue) > 1000000
ORDER BY totalsale DESC
```



Another Helpful SQL Tool

Want to translate a value or set of values?

Use the T-SQL CASE expression

```
SELECT CASE expression
      WHEN expression_value THEN expression_result
      {WHEN expression_value2 THEN expression_result2}
      {ELSE expression_else}
END
FROM ...
```

- Evaluates each WHEN expression value and if it matches shows the expression result
- Otherwise shows the ELSE condition. If no ELSE condition then *NULL*



CASE Example

Q: In the AdventureWorks2017 database, how do we display the “MaritalStatus” column as more than a single character code when retrieving employee information?

```
USE Adventureworks2017
GO
SELECT e.BusinessEntityID AS employeeID,
       CASE e.MaritalStatus
           WHEN 'S' THEN 'Single'
           WHEN 'M' THEN 'Married'
           WHEN 'D' THEN 'Divorced'
           ELSE 'Unknown: ' + e.MaritalStatus
       END AS maritalStatus
FROM HumanResources.Employee AS e
```



Try This CASE!

Q: In the AdventureWorks2017 database how would you show the employee ID and whether they are a salaried employee (hint: SalariedFlag column) as a “Yes” or “No” value?

```
USE Adventureworks2017
GO
SELECT e.BusinessEntityID AS employeeID,
       CASE e.SalariedFlag
           WHEN 1 THEN 'Yes'
           WHEN 0 THEN 'No'
       END AS salariedStatus
  FROM HumanResources.Employee AS e
```



Try Another CASE!

Q: In the SIS database get the student numbers for CourseOfferingID of 19 and show their finalMark along with a grade letter corresponding to the following translation:

A= 90%+ / B= 80%-89% / C= 66%-79% / D= 55%-655 / F=less than 55%

```
USE SIS
GO
SELECT cs.studentNumber, cs.finalMark,
CASE WHEN cs.finalMark >= 90 THEN 'A'
      WHEN cs.finalMark BETWEEN 80 AND 89 THEN 'B'
      WHEN cs.finalMark BETWEEN 66 AND 79 THEN 'C'
      WHEN cs.finalMark BETWEEN 55 AND 65 THEN 'D'
      ELSE 'F'
END AS finalGrade
FROM dbo.CourseStudent AS cs
WHERE cs.CourseOfferingId = 19
```



Subqueries

A subquery is simply a query within another query

- Also called an *inner query* or *inner select*
- Statement containing the subquery is called an *outer query* or *outer select*
- Many subqueries *can* be re-written as regular joins
- Can be used in the SELECT, FROM or WHERE clause
- Used more frequently with tables with a large volume of data



Subquery Syntax - SELECT

Used in a SELECT statement, the subquery must return one and only one row

- Otherwise it will result in an error

```
SELECT table1.column,  
      ( SELECT expression  
          FROM table2  
          WHERE conditions ) AS subquery  
      FROM table1...  
      WHERE conditions...
```



Subquery Syntax - FROM

Used in a FROM statement, the subquery is called an *inline view*
– Any columns in subquery can be referenced

```
SELECT table1.column, subquery.column
FROM table1,
    ( SELECT expression
        FROM table2
        WHERE conditions ) AS subquery
WHERE table1.column = subquery.column
```



Subquery Syntax - WHERE

Used in a WHERE statement, the subquery is called a *nested subquery*

- Need to take care of way the column is joined to the subquery

```
SELECT table1.column, ...
      FROM table1,
WHERE table1.column <operator> ( SELECT expression
                                FROM table2
                                WHERE conditions ) AS subquery
```

- If the subquery returns a single value, use =, !=, <, <=, >, >=
- If the subquery returns more than one value, use IN or NOT IN



Subquery Example

What are the course grades for students who have committed Type “A” academic offences?

```
SELECT studentNumber  
FROM dbo.StudentOffence  
WHERE penaltyCode = 'A'
```



```
SELECT studentNumber, finalMark  
FROM dbo.CourseStudent
```

The search condition (WHERE clause) is evaluated for each row in the CourseStudent table.

If the studentNumber column matches any computed by the subquery, then the WHERE condition evaluates to TRUE.



Subquery Comparison Operators

Q: What are the course grades for students who have not committed a Type “A” academic offense?

```
USE SIS
GO
SELECT studentNumber, finalMark
    FROM dbo.CourseStudent AS cs
   WHERE cs.studentNumber != (SELECT studentNumber
                               FROM dbo.Studentoffence AS so
                              WHERE so.penaltyCode = 'A' )
```

BUT what if there were two or more students in the table who had committed a Type “A” academic offense?

- Need to use NOT IN rather than !=



Subquery IN/NOT IN Operator

Q: What are the course grades for students who have not committed a Type “A” academic offense?



Subquery IN/NOT IN Operator - II

Use IN and NOT IN to compare the results of subquery that returns multiple rows

```
SELECT ...
  FROM ...
 WHERE column IN ( SELECT ... )

SELECT ...
  FROM ...
 WHERE column NOT IN ( SELECT ... )
```



Subqueries – Try This!

Q: Using a subquery, get the names of the employees who work in the School of Business (Hint: Tables Person and Employee)

```
USE SIS
GO
SELECT p.lastName + ', ' + p.firstName AS employeeName
    FROM dbo.Person AS p
 WHERE p.number IN ( SELECT e.number
                        FROM dbo.Employee AS e
                      WHERE e.schoolCode = 'BUS' )
```

- Note that within SSMS you can highlight part of a query to execute it
 - Try this with the subquery in this example

Subquery - Existence

You can also use the EXISTS and NOT EXISTS keywords to evaluate the contents of a subquery for data existence

```
SELECT ...
  FROM table1
WHERE EXISTS ( SELECT ...
                  FROM table2
                  WHERE table2.column = table1.column )

SELECT ...
WHERE NOT EXISTS ( SELECT ...
                      FROM table2
                      WHERE table2.column = table1.column )
```



Subquery Existence Example

Q: Using a subquery, get the names of the employees who work in the School of Business (Hint: Tables Person and Employee)

```
USE SIS
GO
SELECT p.lastName + ', ' + p.firstName
    FROM dbo.Person AS p
 WHERE EXISTS ( SELECT *
                  FROM dbo.Employee AS e
                 WHERE e.number = p.number
                   AND e.schoolCode = 'BUS' )
```



- Same result-set, just a different approach to the problem
- Column in the inner query references the column in the outer query



A Milestone Moment!

We have covered

- Database Concepts
- Database Data Types
- ***Many*** Methods of Data Retrieval

You now have the tools to
retrieve data from complex table
structures!

Congratulations!



Practice Time!

- A. In SIS, get a list of the *unique* student numbers and whether they are international students from the dbo.Audit table. Convert the numeric indicator for international students to a “Yes” or “No” result. Sort the results by student number in descending order. (Hint: Use CASE)
- B. In AdventureWorks2017, get the Sales Person ID and the total of their sales from the SalesOrderHeader table. Order them in descending order by total sales. Implement this by using a subquery in the SELECT portion of the query.
- C. In AdventureWorks2017, get a list of currencies from the Sales.Currency table that are not used in any country as defined in the Sales.CountryRegionCurrency table. Structure your SQL using a subquery in your WHERE clause with the NOT IN operation.
- D. Create a query for the same result-set as in question B. above but by using NOT EXISTS with the subquery instead.
- E. Create a query to get a list of customer IDs and Account Numbers who are in the Sales.Customer table in AdventureWorks 2017 who do not have any purchases in the SalesOrderHeader table. Use a subquery (flavour of your choice) to get your results.



Summary

- Review Session Concepts
- This Week's Assignment
- Current Assignment Reminder
- Questions?



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

Fall 2019 - Week 5 – Thursday Lesson



Agenda

- Daily Check-In / Administrivia
- Terminology/Topic Review
- Last Week's Questions Review
- “CRUD” Operations
- Transactions
- Practice!
- Summary



The Daily Dose

- Thumb-Ball Time!
- Check-In With Everyone
- Current Assignment Status
- Mid-Term Information
- Use www.menti.com to ask me anything
 - Code is on the board



Fun “Data” of the Day

Source: <https://www.livin3.com/50-cool-and-weird-fun-facts-that-you-should-know>

Donald duck comics were banned from Finland because he doesn't wear pants

Pirates wore earrings because they believed it improved their eyesight



Recent Terminology & Topics

- The T-SQL CASE Statement
- Subqueries
 - SELECT
 - FROM
 - WHERE
- Subquery Operators
 - =/!=/etc. vs. IN/NOT IN/EXISTS/NOT EXISTS
- Our Big Milestone!





SQL Statement Anatomy

```
SELECT [DISTINCT] {*/table1.column/table2.column/expression} {AS alias} [,...]
    FROM table1
{INNER/LEFT/RIGHT/FULL} JOIN table2 ON table2.column = table1.column
    WHERE conditions
{GROUP BY non-aggregate expression(s)}
{HAVING aggregate expression condition(s)}
    ORDER BY column [ASC/DESC] [,...]
        {OFFSET offsetNumRows {ROW|ROWS}}
        {FETCH {FIRST|NEXT} fetchNumRows {ROW|ROWS} ONLY}
```



Last Week's Questions Review - I

- A. In SIS, get a list of the *unique* student numbers and whether they are international students from the dbo.Audit table. Convert the numeric indicator for international students to a “Yes” or “No” result. Sort the results by student number in descending order. (Hint: Use CASE)

```
USE SIS
GO
SELECT DISTINCT a.studentNumber,
               CASE a.isInternational
                   WHEN 0 THEN 'No'
                   WHEN 1 THEN 'Yes'
               END AS internationalStudent
        FROM dbo.Audit AS a
   ORDER BY a.studentNumber DESC
```



Last Week's Questions Review - II

- B. In AdventureWorks2017, get the Sales Person ID and the total of their sales from the SalesOrderHeader table. Order them in descending order by total sales. Implement this by using a subquery in the SELECT portion of the query.

```
USE Adventureworks2017
GO
SELECT DISTINCT s.SalesPersonID,
    ( SELECT SUM( salesSub.TotalDue )
        FROM Sales.SalesOrderHeader AS salesSub
        WHERE salesSub.SalesPersonID = s.SalesPersonID ) AS totalsales
    FROM Sales.SalesOrderHeader AS s
ORDER BY totalsales DESC
```



Last Week's Questions Review - III

- C. In AdventureWorks2017, get a list of currencies from the Sales.Currency table that are not used in any country as defined in the Sales.CountryRegionCurrency table. Structure your SQL using a subquery in your WHERE clause with the NOT IN operation.

```
USE Adventureworks2017
GO
SELECT c.CurrencyCode, c.Name
  FROM Sales.Currency AS c
 WHERE c.CurrencyCode NOT IN ( SELECT crc.CurrencyCode
                                FROM Sales.CountryRegionCurrency AS crc )
```



Last Week's Questions Review - IV

- D. Create a query for the same result-set as in question C above but by using NOT EXISTS with the subquery instead.

```
USE Adventureworks2017
GO
SELECT c.CurrencyCode, c.Name
    FROM Sales.Currency AS c
 WHERE NOT EXISTS ( SELECT *
        FROM Sales.CountryRegionCurrency AS crc
        WHERE crc.CurrencyCode = c.CurrencyCode )
```

Last Week's Questions Review - V

- E. Create a query to get a list of customer IDs and Account Numbers who are in the Sales.Customer table in AdventureWorks 2017 who do not have any purchases in the SalesOrderHeader table. Use a subquery (flavour of your choice) to get your results.

“CRUD” Operations

Create

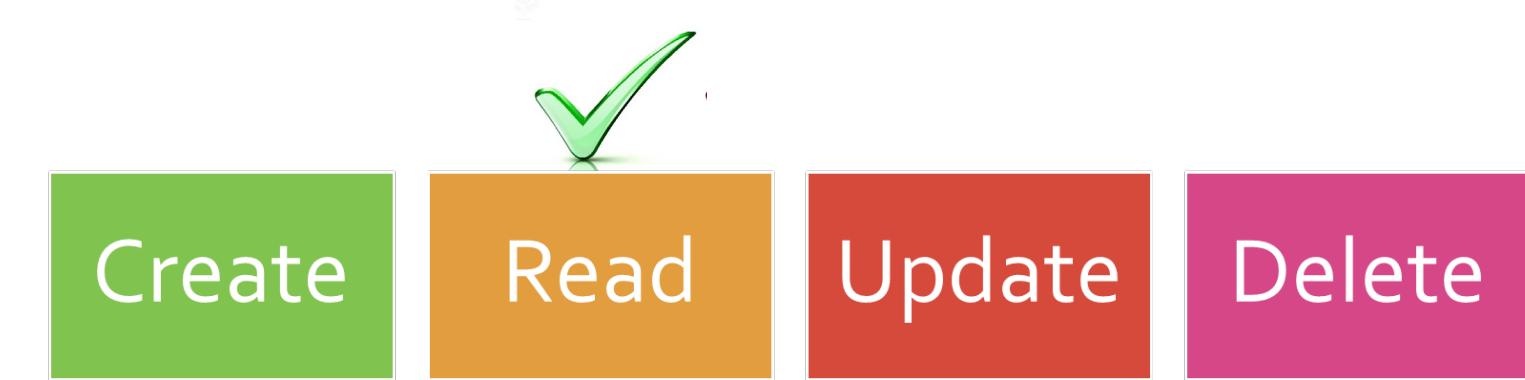
Read

Update

Delete

In computer programming, the acronym CRUD stands for Create, Read, Update and Delete. They are the four basic operations done in any data repository.

“CRUD” Operations



We've already tackled “Read” with SQL SELECT statements.

Much of what we've learned (forming the WHERE clause) carries over into the other operations



CRUD - UPDATE

Allows us to update data in a database table

- Syntax:

```
UPDATE table
    SET column1 = value1,
        column2 = value2 [...]
    WHERE condition(s)
```

Note:

- You *can* update columns that you use in the WHERE clause
- The statement is **atomic**; either all of the intended rows are updated or the entire UPDATE statement is undone in error
- **Take CARE** with the WHERE clause to use key information

UPDATE Example

Q: In the SIS database, we need to update Barbara Fousek's first name to "Barb". How do we do it?

- First, find Barbara Fousek's record in the dbo.Person table:

```
SELECT *
  FROM dbo.Person
 WHERE lastName = 'FOUSEK'
   AND firstName = 'BARBARA'
```

- From this we can get the *Primary Key* for this record. Her "number" is 5715438
- That makes the update look like this:

```
UPDATE dbo.Person
 SET firstName = 'BARB'
 WHERE number = 5715438
```

UPDATE Example Notes

Q: In the SIS database, we need to update Barbara Fousek's first name to "Barb". How do we do it?

- We could also do this:

```
UPDATE dbo.Person  
    SET firstName = 'BARB'  
  WHERE lastName = 'FOUSEK'  
        AND firstName = 'BARBARA'
```

- BUT, what if there were multiple Barbara Fousek records?
- In most-cases this is probably fine but it's always good to verify the data and double-check. Be *sure* of the data.



UPDATE Try This #1

Q: In SIS, the course PROG8080 in the dbo.CourseOffering table needs to be renamed to PROG8081. How do we do it?

```
UPDATE dbo.CourseOffering
    SET courseNumber = 'PROG8081'
 WHERE courseNumber = 'PROG8080'
```

UPDATE Try This #2

Q: In SIS, update the CourseOffering capacity to 44 where the course has a CoursePrerequisiteAnd of “PROG1780”.

- First, look at the data:

```
SELECT *
  FROM dbo.CourseOffering AS co
 WHERE co.courseNumber IN ( SELECT cpa.courseNumber
                            FROM dbo.CoursePrerequisiteAnd cpa
                            WHERE cpa.prerequisiteNumber = 'PROG1780' )
 ORDER BY co.courseNumber
```

- Note the rows
- Update:

```
UPDATE dbo.CourseOffering
      SET capacity = 44
 WHERE courseNumber IN ( SELECT courseNumber
                           FROM dbo.CoursePrerequisiteAnd
                           WHERE prerequisiteNumber = 'PROG1780' )
```

UPDATE Notes

- UPDATE statements will update **ALL** of the rows that match the criteria provided
 - So if you don't provide a WHERE clause, ALL rows in the table will be updated
- To check what will be updated, use a SELECT statement with the same criteria
- The results of the UPDATE are **IMMEDIATE** and cannot be un-done unless you contain your statement within a **TRANSACTION** (What's that you ask?)



TRANSACTIONS

A transaction is a set of T-SQL statements that are executed together as a unit

- If successful, the data is *committed* and the changes made by the statements are permanently saved to the database
- User has the opportunity to *rollback* the changes so that nothing is applied to the database
- Basic syntax:

```
BEGIN { TRAN | TRANSACTION }
-- SQL statements here
{COMMIT | ROLLBACK} [ {TRAN | TRANSACTION} ]
```



TRANSACTION Examples

The following SQL shows a transaction with two updates that are then committed to the database:

```
BEGIN TRANSACTION
    UPDATE dbo.Person SET firstName = 'Al';
    UPDATE dbo.Person SET lastName = 'Everyone';
COMMIT
```

The following SQL shows a transaction with two updates that are then rolledback from the database:

```
BEGIN TRANSACTION
    UPDATE dbo.Person SET firstName = 'Mary';
    UPDATE dbo.Person SET lastName = 'Queen';
ROLLBACK
```



TRANSACTION Notes

TRANSACTION logic is very handy when you're working in your database. Consider running the following in sequence:

```
SELECT MAX(capacity)  
      FROM dbo.CourseOffering  
-- This should return a value of 30
```

```
BEGIN TRANSACTION  
UPDATE dbo.CourseOffering  
SET capacity = 99
```

```
SELECT MAX(capacity)  
      FROM dbo.CourseOffering  
-- This should return a value of 99
```

ROLLBACK

```
SELECT MAX(capacity)  
      FROM dbo.CourseOffering  
-- This should return the original value of 30
```

“CRUD” Operations

Create



Read



Update

Delete

We've already tackled “Read” with SQL
SELECT statements.

We just learned how to “Update” tables with
SQL UPDATE statements



CRUD - DELETE

Allows us to delete data in a database table

- Syntax:

```
DELETE
    FROM table
 WHERE condition(s)
```

Note:

- The statement is **atomic**; either all of the intended rows are deleted or the entire DELETE statement is undone in error
- **Take CARE** with the WHERE clause to use key information

DELETE Example

Q: In the SIS database, we need to delete Barbara Fousek's Person record. How do we do it?

- First, find Barbara Fousek's record in the dbo.Person table:

```
SELECT *
  FROM dbo.Person
 WHERE lastName = 'FOUSEK'
   AND firstName = 'BARBARA'
```

- From this we can get the *Primary Key* for this record. Her “number” is 5715438
- That makes the delete look like this:

```
DELETE
  FROM dbo.Person
 WHERE number = 5715438
```



DELETE Example Notes

Q: In the SIS database, we need to delete Barbara Fousek's Person record. How do we do it?

- We could also do this:

```
DELETE
  FROM WHERE lastName = 'FOUSEK'
        AND firstName = 'BARBARA'
```

- BUT what if there were multiple Barbara Fousek records?
- In most-cases this is probably fine but it's always good to verify the data and double-check. Be *sure* of the data.



DELETE Try This #1

Q: In SIS, the course PROG8090 in the dbo.CourseOffering table needs to be removed. How do we do it?

```
DELETE
    FROM dbo.CourseOffering
    WHERE courseNumber = 'PROG8090'
```



DELETE Try This #2

Q: Employee Number 7572373 is to be removed from the course offering table. How do we do that?

- First, look at the data:

```
SELECT *
  FROM dbo.CourseOffering AS co
 WHERE co.employeeNumber = 7572373
```

- Note the rows
- Delete:

```
DELETE
  FROM dbo.CourseOffering
 WHERE employeeNumber = 7572373
```

DELETE Notes

- DELETE statements will update **ALL** of the rows that match the criteria provided
 - So if you don't provide a WHERE clause, ALL rows in the table will be removed
- To check what will be updated, use a SELECT statement with the same criteria
- The results of the DELETE are **IMMEDIATE** and cannot be un-done unless you contain your statement within a **TRANSACTION** statement (Which you now know about)

TRUNCATE TABLE

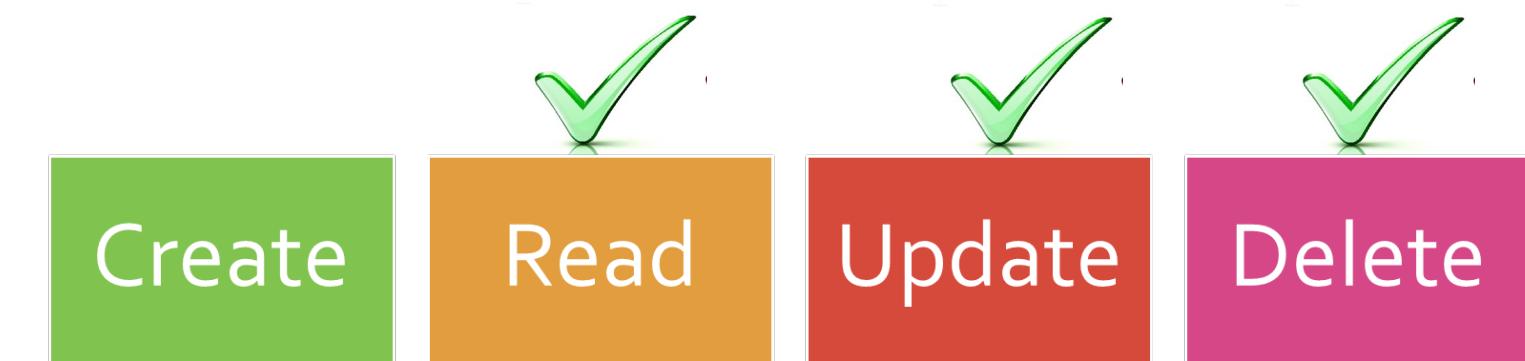
Many databases and the ISO SQL Standard include a statement to allow you to remove *every row* in a table

- Syntax: `TRUNCATE TABLE table`

Notes:

- You cannot provide criteria to only delete certain rows. It's the entire table contents that are removed
- It is faster to TRUNCATE the table than to DELETE the entire contents (TRUNCATE is not logged)
- A truncated table *cannot* be rolled back in a transaction

“CRUD” Operations



We've already tackled “Read” with SQL SELECT statements.

We learned how to “Update” tables with SQL UPDATE statements

We just learned how to “Delete” tables with SQL DELETE and TRUNCATE statements

General Notes

Now that we're updating data in the database, you may need to restore a database to its original state for assignments and tests

- To delete the database, right click on it in the Object Explorer in SSMS and select “Delete”
 - BE SURE to click the “Close existing connections” checkbox before hitting OK
- Re-import or create the database using the backup file or SQL script like we did in the second class

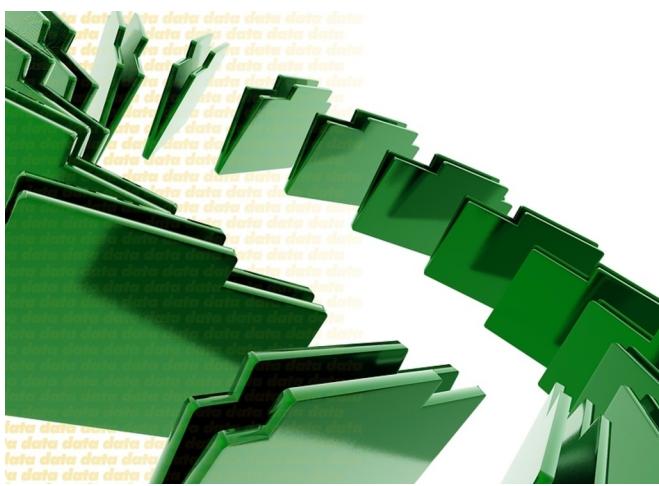
Practice Time!

- A. In SIS, update the street address of Student Number 2353688 to 220 First Street, Waterloo, ON N2K 5G7.
- B. In SIS, update the ProgramFee table so that the *international* tuition for Program Code “0057C” for any semester is \$1999.
- C. In SIS within a transaction update the ProgramFee table to make tuition free (\$0) for Program Code “1009”. Do not commit the transaction but select the information for Program Code 1009 after the update, see that it has been changed then rollback the transaction and run your select again to see that it was not updated
- D. In SIS, delete every person from the dbo.Person table who’s first name is “Mark”. Make sure to include any who have middle initials in the firstName column. Do this within a transaction that you commit at the end.
- E. In SIS, delete the entire contents of the dbo.StudentOffence table.
- F. Delete the entire SIS database and then recreate it so that we’re back to a fresh database.



Summary

- Review Session Concepts
- This Week's Assignment
- Current Assignment Reminder
- Questions?



WEEK #6

ASSIGNMENT

In-Class Part of 10% Individual Assignment

Half-Way There Assignment

Mark Morell

Database Management – Fall 2019

Assignment Type:

- Individual – Prepare and submit your results independently

Date Due:

- Thursday, October 10th IN-CLASS

Instructions:

- Please submit your assignment electronically through eConestoga. This part **may not** be altered or resubmitted after the assignment time in class has ended
- Assignments should be submitted as Microsoft Word files using the course coversheet format. You **MUST** include your query as text in the Word document as well as a **FULL screenshot** of your SSMS screen (screen capture the entire application screen including the title bar through the bottom of the window). Multiple screenshots may be required.
- Please include the question number with your responses.

Late Assignment Penalty:

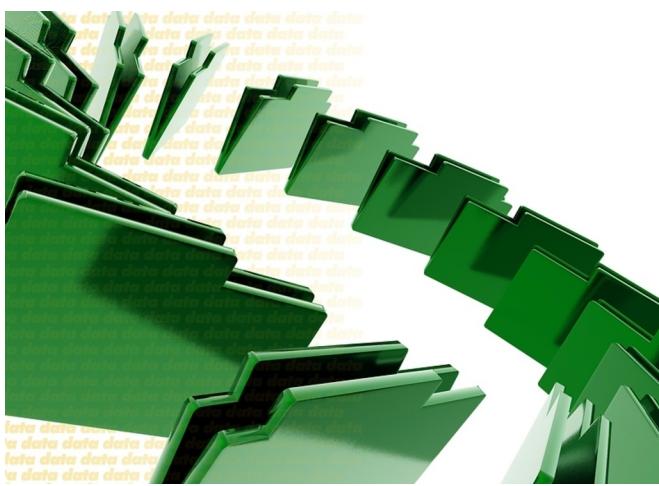
- This in-class assignment must be done in the allotted time in class. It cannot be submitted late

Assignment Questions

NOTE: Use the Costume database (installed below) to answer *all* of these questions. Be sure to comment all of your queries where asked to describe what you are doing.

Question #	Question	Score
1	Download the CreateCostumeDatabase.sql file from this week's eConestoga information. Run the script to create the database and take an image of SSMS with Object Explorer open, the Costume database expanded and the tables entry expanded to show the list of tables successfully created.	1
2	Write a T-SQL query that shows the total number (i.e. a number, not the list of data) of costumes that are of a colour that starts with the letter "B"	2
3	<p>Write a single query that retrieves a list of costumes that are either "Not Scary" or "Slightly Frightening". Include the following in your results:</p> <ul style="list-style-type: none"> • Costume Name • Costume Description • Costume Colour • Scare Rating Description <p>In the single query, order your results first by scare rating description in ascending order and then secondarily in descending order by costume name.</p>	3
4	<p>Write a single query that retrieves a list of costumes whose CostumeInventory record shows that they've been lost <u>more times</u> than the number that are in-stock for that costume. Show the following in the results:</p> <ul style="list-style-type: none"> • Costume Name • Number In Stock • Number Lost • The difference between the number lost and those in stock. Give this column the name "LostExceedsInStockBy" <p>Choose how to sort your results and include a comment in the SQL code saying why you chose to sort the way that you did.</p>	4
5	<p>Doug Graves has returned his Dracula costume and it has not been damaged. Within a <u>transaction</u>, update the rental information with the current date/time as the returned date and indicate that it has not been damaged. With your submission show your SQL statement and also SELECT the data from the table showing that it has been updated. Be sure to end your transaction so that the data is written to the database.</p>	3
6	<p>Write 2 SQL statements as follows:</p> <ol style="list-style-type: none"> 1. The first will create a brand new costume in the Costume Table. Populate the information related to the costume as you see fit with something fun! 2. The second will create an entry in the CostumeInventory table for your new costume. Populate the data as you'd like. <p>After executing your queries, select the new data from both tables and include an image of it (along with your creation SQL and image).</p>	4
7	<p>Write a SQL statement that will create a copy of the structure and contents of the Costume table. Name it "CostumeBackup". Take an image to show the execution of your SQL. After creation write a SQL statement to select the contents of the new table and provide an image of the results.</p>	2

8	Write a SQL statement to completely delete the contents of the new backup table created in the question above. After executing the statement create a SQL statement to select the contents of the table and include the results in your submission.	1
	Total	20



WEEK #6

ASSIGNMENT

Part of 10% Individual Assignment

Half-Way There Assignment

Mark Morell

Database Management – Fall 2019

Assignment Type:

- Individual – Prepare and submit your results independently

Date Due:

- Thursday, October 17th by the end of the day

Instructions:

- Please submit your assignment electronically through eConestoga.
- Assignments should be submitted as Microsoft Word files using the course coversheet format. You **MUST** include your query as text in the Word document as well as a **FULL screenshot** of your SSMS screen (screen capture the entire application screen including the title bar through the bottom of the window). Multiple screenshots may be required.
- If you are using external sources (images, text, etc.) you must reference them as part of your assignment and not copy them as-is.
- Best practice is to research your answers and then write the response to the question in your own words.
- Please include the question number with your responses.

Late Assignment Penalty:

Days Late	Penalty %
1	5
2	10
3	20
4	40
5	60
6	80
7	100

Assignment Questions

NOTE: Use the Costume database to answer *all* of these questions

Question #	Question	Score																									
1	<p>Write a single query to show a list of costumes that have been rented within the last 2 weeks. In your query include the following information:</p> <ul style="list-style-type: none"> • Costume rented • Name of person who rented the costume (formatted nicely) • Date rented (just the date without a time) <p>Sort the result with the latest rental first</p>	4																									
2	Describe what NULL represents related to databases. Compare it to other possible values that could be used in a database.	4																									
3	<p>What T-SQL data type would you use to store the following information and why would you use it?</p> <ol style="list-style-type: none"> a) A value that can be either true or false b) A value that's a very large number used for scientific purposes c) The age of a person d) The cost of items at a supermarket e) Values to represent when the next solar eclipse will occur f) A list of ingredients for pumpkin pie g) A Word document 	7																									
4	<p>Answer these questions using the following partial table of dog breeds:</p> <table border="1"> <thead> <tr> <th>id</th> <th>name</th> <th>size</th> <th>colour</th> <th>origin</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Labrador Retriever</td> <td>Large</td> <td>Black, Brown, Golden</td> <td>Canada, UK</td> </tr> <tr> <td>2</td> <td>Bulldog</td> <td>Medium</td> <td>Brown, White</td> <td>UK</td> </tr> <tr> <td>3</td> <td>German Shepherd</td> <td>Large</td> <td>Brown, Black</td> <td>Germany</td> </tr> <tr> <td>4</td> <td>Rajapalayam</td> <td>Large</td> <td>White</td> <td>India</td> </tr> </tbody> </table> <ol style="list-style-type: none"> a) What are the various candidate keys for this table? b) Are each of these keys natural keys or surrogate keys? c) What would make the best choice for the primary key and why? 	id	name	size	colour	origin	1	Labrador Retriever	Large	Black, Brown, Golden	Canada, UK	2	Bulldog	Medium	Brown, White	UK	3	German Shepherd	Large	Brown, Black	Germany	4	Rajapalayam	Large	White	India	6
id	name	size	colour	origin																							
1	Labrador Retriever	Large	Black, Brown, Golden	Canada, UK																							
2	Bulldog	Medium	Brown, White	UK																							
3	German Shepherd	Large	Brown, Black	Germany																							
4	Rajapalayam	Large	White	India																							
5	Describe what a database foreign key is and give a real-life example of one.	4																									
6	<p>What will the following query retrieve?</p> <pre>SELECT costumeName FROM dbo.Costume WHERE id IN (SELECT costumeId FROM dbo.CostumeRental WHERE returnedDate IS NOT NULL AND returnDamaged = 1)</pre>	4																									
7	<p>Re-write the following query using an IN statement:</p> <pre>SELECT costumeName FROM dbo.Costume WHERE colour = 'Brown' OR colour = 'Black' OR colour = 'Green' OR colour = 'White'</pre>	2																									

8	<p>Write a query to retrieve the <u>top 60%</u> of all costumes that have a mask in their costume description. Provide the following information:</p> <ul style="list-style-type: none"> • Costume Name • Costume Description • Costume Colour • Number of pieces • Number currently in stock • The date it was last rented <p>Order the single set of results by costume name. Note that there may be costumes that have not been rented so take that into account</p>	6
9	Describe 5 traits of good SQL coding style	5
10	Write a query that simply gives us (from a single query) the total number of costumes that are in-stock, the total number that are on-order and the total number that have been lost. Be sure to include appropriate column heading names in your query.	3
11	Write a query to update the costume rental information. Rachel Scruff just now returned the costume that she had rented and it is damaged. Include a comment that she claims it was damaged before she rented it. In your results show your update statement and then write a SELECT statement to show that the data has been updated in the table.	4
12	We need to remove all of the data from a table in the database that has lots and lots of data. What's the quickest way to clear it all out so that no rollback log is created?	2
13	Write a single query that retrieves the costumes names and how many times that they've been rented as shown in the CostumeRental table. Sort the results by the number of times that they've been rented and then secondarily within the same query by costume name. Note that this is a bit tricky so a) Start building your SQL in steps to help get you what you need and b) don't forget to include costumes that have never been rented. This should result in 12 rows being returned.	6
14	<p>Write a query that would create a brand new table of costumes named "ScaryCostumes" that include only those that are rated at "Scary" or above on the scare-factor. Include the following information in the new table:</p> <ul style="list-style-type: none"> • Costume Name • Costume Description • UPC Code 	3
	Total	60
	Total From In-Class Component	20
	Grand Total	80



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

Fall 2019 - Week 6 – Monday Lesson



Agenda

- Daily Check-In / Administrivia
- Terminology/Topic Review
- Last Week's Questions Review
- More “CRUD” Operations - INSERT
- Practice!
- Summary



The Daily Dose

- Thumb-Ball Time!
- Check-In With Everyone
- Current Assignment Status
- Mid-Term Information
- Use www.menti.com to ask me anything
 - Code is on the board



Fun “Data” of the Day

Source: <https://www.livin3.com/50-cool-and-weird-fun-facts-that-you-should-know>

If you started with \$0.01 and doubled your money every day, it would take 27 days to become a millionaire

You can't hum while holding your nose closed



Recent Terminology & Topics

- CRUD Operations
 - Create = ?
 - Read = SELECT
 - Update = UPDATE
 - Delete = DELETE
- Transactions





SQL Statement Anatomy

```
SELECT [DISTINCT] {*/table1.column/table2.column/expression} {AS alias} [,...]
    FROM table1
{INNER/LEFT/RIGHT/FULL} JOIN table2 ON table2.column = table1.column
    WHERE conditions
{GROUP BY non-aggregate expression(s)}
{HAVING aggregate expression condition(s)}
    ORDER BY column [ASC/DESC] [,...]
        {OFFSET offsetNumRows {ROW|ROWS}}
        {FETCH {FIRST|NEXT} fetchNumRows {ROW|ROWS} ONLY}
```



Last Week's Questions Review - I

- A. In SIS, update the street address of Student Number 2353688 to:
220 First Street, Waterloo, ON N2K 5G7

```
USE SIS
GO
UPDATE dbo.Student
    SET localStreet = '220 First Street',
        localCity = 'waterloo',
        localProvinceCode = 'ON',
        localPostalCode = 'N2K 5G7'
    WHERE number = 2353688
```



Last Week's Questions Review - II

- B. In SIS, update the ProgramFee table so that the *international* tuition for Program Code “0057C” for any semester is \$1999.

```
USE SIS
GO
UPDATE dbo.ProgramFee
    SET internationalTuition = 1999
WHERE code = '0057C'
```



Last Week's Questions Review - III

- C. In SIS within a transaction update the ProgramFee table to make tuition free (\$0) for Program Code "1009". Do not commit the transaction but select the information for Program Code 1009 after the update, see that it has been changed then rollback the transaction and run your select again to see that it was not updated

```
USE SIS
GO
BEGIN TRANSACTION
UPDATE dbo.ProgramFee
    SET tuition = 0
WHERE code = '1009'

SELECT * FROM dbo.ProgramFee WHERE code = '1009'

ROLLBACK

SELECT * FROM dbo.ProgramFee WHERE code = '1009'
```



Last Week's Questions Review - IV

- D. In SIS, delete every person from the dbo.Person table who's first name starts with "Mark". Do this within a transaction that you commit at the end.

```
USE SIS
GO
BEGIN TRANSACTION

DELETE FROM dbo.Person WHERE firstName LIKE 'MARK%'

COMMIT
```

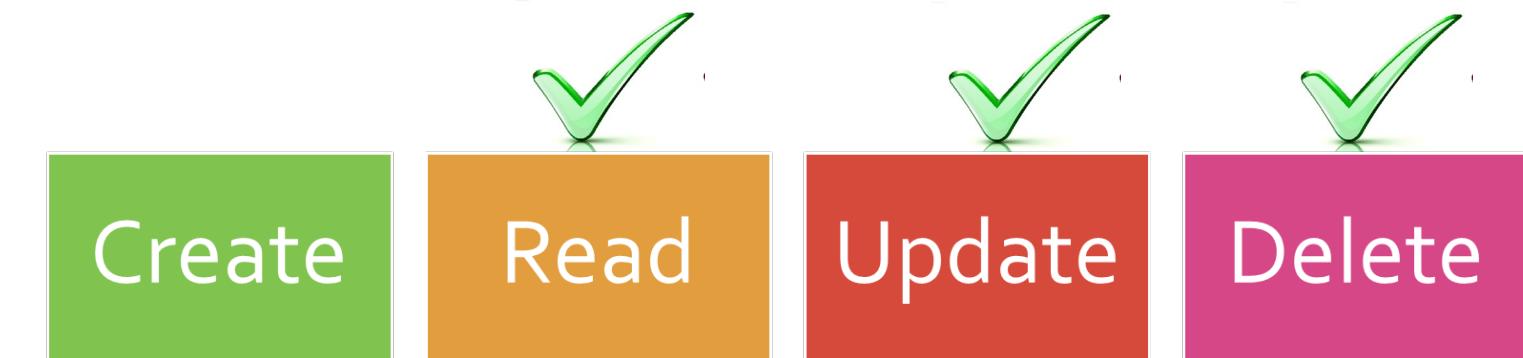


Last Week's Questions Review - V

- E. In SIS, delete the entire contents of the dbo.StudentOffence table.

```
USE SIS
GO
TRUNCATE TABLE dbo.StudentOffence
```

“CRUD” Operations



We've already tackled “Read” with SQL SELECT statements.

We learned how to “Update” tables with SQL UPDATE statements

We just learned how to “Delete” tables with SQL DELETE and TRUNCATE statements



CRUD - INSERT

Allows us to insert new data in a database table

- General Syntax:

```
INSERT INTO table (column1, column2, ..., columnx)
VALUES (value1, value2, ..., valuex)
```

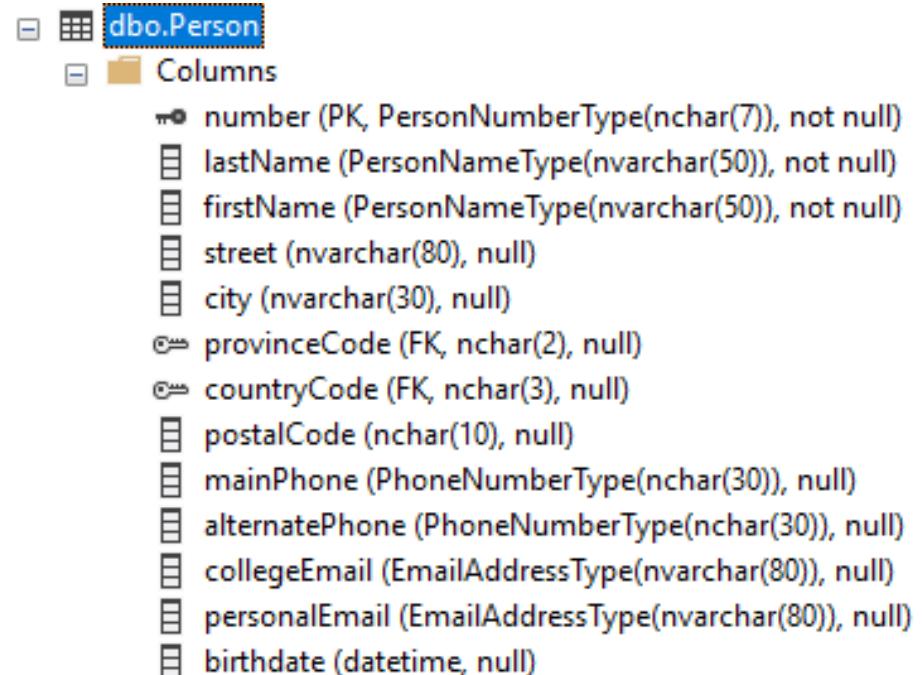
Notes:

- You can leave the column names off of the INSERT statement
IF you provide values for every single column
 - If you aren't inserting into every column you **MUST** list them
 - Column values not provided will be inserted as NULL



INSERT Example

Q: In the SIS database, we need to add a new person to the dbo.Person table. How do we do it? Use a SELECT to check your results.



```
INSERT INTO dbo.Person ( number, lastName, firstName, street, city, provinceCode, countryCode, postalCode, mainPhone, alternatePhone, collegeEmail, personalEmail, birthdate )
VALUES ( 999999, 'BING', 'CHANDLER', '123 Friend Street', 'New York', 'NY', 'USA', '66502',
        '(555)123-4567', NULL, 'cbing@conestogac.on.ca', 'thebing@bing.com', '1968-04-08' )
```



INSERT Example – Another Way

Q: In the SIS database,
we need to add a new
person to the dbo.Person
table.

dbo.Person	
	Columns
PK	number (PK, PersonNumberType(nchar(7)), not null)
	lastName (PersonNameType(nvarchar(50)), not null)
	firstName (PersonNameType(nvarchar(50)), not null)
	street (nvarchar(80), null)
	city (nvarchar(30), null)
FK	provinceCode (FK, nchar(2), null)
FK	countryCode (FK, nchar(3), null)
	postalCode (nchar(10), null)
	mainPhone (PhoneNumberType(nchar(30)), null)
	alternatePhone (PhoneNumberType(nchar(30)), null)
	collegeEmail (EmailAddressType(nvarchar(80)), null)
	personalEmail (EmailAddressType(nvarchar(80)), null)
	birthdate (datetime, null)

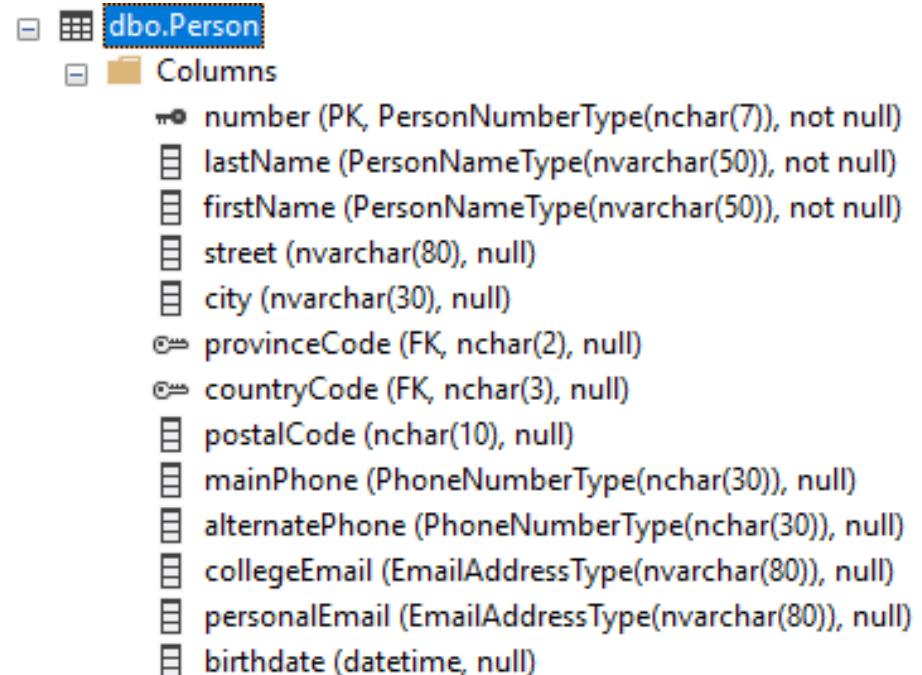
```
INSERT INTO dbo.Person      -- NOTE: We are omitting the column list
VALUES ( 999999, 'BING', 'CHANDLER', '123 Friend Street', 'New York', 'NY', 'USA', '66502',
        '(555)123-4567', NULL, 'cbing@conestogac.on.ca', 'thebing@bing.com', '1968-04-08' )

-- This works because we provided ALL of the columns including a NULL where we didn't
-- have a value to insert
```



INSERT Example II

Q: In the SIS database, we need to add a new person to the dbo.Person table but we don't know much information. How do we do it?



```
INSERT INTO dbo.Person ( number, lastName, firstName, city, provinceCode, countryCode,
collegeEmail )
VALUES ( 999998, 'GREEN', 'RACHEL', 'New York', 'NY', 'USA', 'rgreen@conestogac.on.ca' )
```

INSERT Example Notes

- To INSERT a record you *must* provide values for all non-NULL columns
 - Otherwise you'll get an “INSERT fails” message
 - The only exception are IDENTITY columns
 - Used by SQL Server as surrogate key columns
 - Uniquely incremented and generated when INSERT is performed
 - You can specifically provide the value b
- TRANSACTION logic applies to INSERT statements too
 - You can COMMIT and ROLLBACK records inserted



INSERT Try This #1

Q: In SIS, INSERT records in the dbo.Person table for Chandler Bing (ID 999999) and Rachel Green (ID 999998).

```
INSERT INTO dbo.Person ( number, lastName, firstName, street, city, provinceCode, countryCode,
    postalCode, mainPhone, alternatePhone, collegeEmail, personalEmail,
    birthdate )
VALUES ( 999999, 'BING', 'CHANDLER', '123 Friend Street', 'New York', 'NY', 'USA', '66502',
    '(555)123-4567', NULL, 'cbing@conestogac.on.ca', 'thebing@bing.com', '1968-04-08' )
```

```
INSERT INTO dbo.Person ( number, lastName, firstName, city, provinceCode, countryCode,
    collegeEmail )
VALUES ( 999998, 'GREEN', 'RACHEL', 'New York', 'NY', 'USA', 'rgreen@conestogac.on.ca' )
```

```
SELECT * FROM dbo.Person WHERE number in (999998, 999999)
```



INSERT Try This #2

Q: In SIS, Chandler and Rachel have become employees at the school. We need to add them to the dbo.Employee table.

```
INSERT INTO dbo.Employee( number, reportsTo, campusCode, schoolCode, location,
                           businessPhone, extension, fax, encryptedSSN )
VALUES( 999998, NULL, 'W', 'BUS', '3G11', '(519)748-5220', '1234', NULL,
        CONVERT(BINARY, 'RachelSSN') )

INSERT INTO dbo.Employee( number, reportsTo, campusCode, schoolCode, location,
                           businessPhone, extension, encryptedSSN )
VALUES( 999999, 999998, 'W', 'BUS', '3G12', '(519)748-5220', '5678',
        CONVERT( BINARY, 'ChandlerSSN' ) )

SELECT * FROM dbo.Employee WHERE number IN (999998, 999999)
```



INSERT from SELECT

Allows us to insert new data in a database table by using a SELECT statement to get the data to insert

- General Syntax:

```
INSERT INTO table (column1, column2, ..., columnx)
SELECT othertable.x, othertable.y, ..., othertable.z
      FROM othertable
        WHERE ...
```

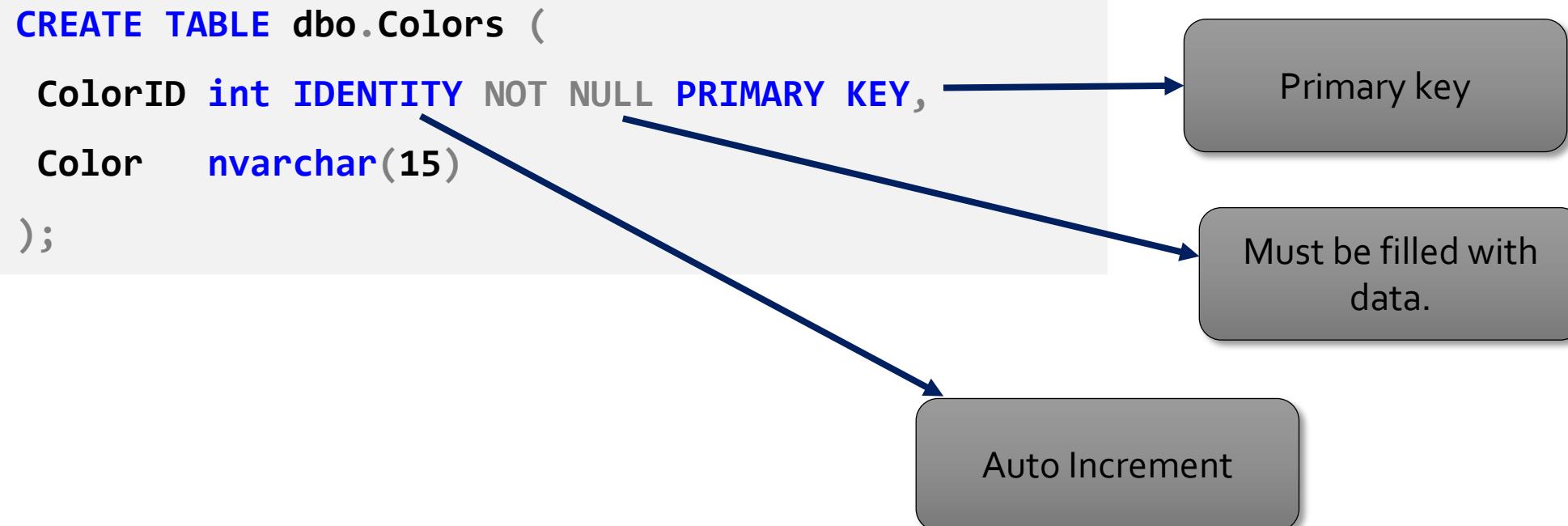
Notes:

- You **must** select the same number of columns or expressions as those in the destination table
- The data types selected must match those in the destination
- It **can** insert multiple rows based on what's retrieved



INSERT from SELECT Example

To demonstrate this let's create a very simple table in the AdventureWorks2017 database:





INSERT from SELECT Example II

To see how the IDENTITY column works try inserting into our new table:

```
INSERT INTO dbo.Colors
VALUES ('Red')

INSERT INTO dbo.Colors
VALUES ('Green')

SELECT * FROM dbo.Colors

DELETE FROM dbo.Colors
```



INSERT from SELECT Example III

Now try to INSERT right into our new table:

```
-- Insert all the unique Product colours into the table:  
  
INSERT INTO dbo.Colors  
SELECT DISTINCT Color  
    FROM Production.Product  
 WHERE Color IS NOT NULL  
  
SELECT * FROM dbo.colors
```



SELECT INTO Statement

Allows you to create a new table populated with the results of the SELECT statement

- General Syntax:

```
SELECT * / columns / expressions
      INTO newtable
      FROM table
      WHERE condition(s)
```

Notes:

- Copies from the source all:
 - Column Names
 - Column Data Types
 - NULL restrictions
 - IDENTITY property
 - Data



SELECT INTO Statement Example

Q: How can we take a subset of Person data and create a new table from it?

```
SELECT LastName, FirstName  
      INTO Person.Person_Backup  
     FROM Person.Person  
    WHERE Title = 'Mr.'  
  
SELECT * FROM Person.Person_Backup
```

Notes:

- It's an easy way to create a temporary or backup table

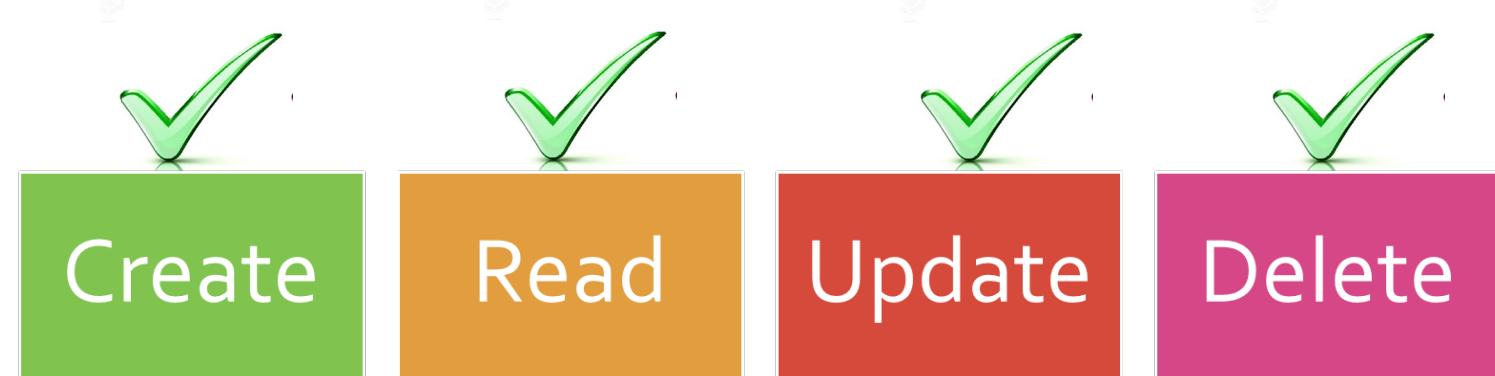


Clean Up!

Q: We've created these temporary tables so what's the syntax to delete them?

DROP TABLE *table*

“CRUD”
Operations
Grand
Finale



We have now learned about all of the
CRUD database operations!

You can INSERT, SELECT, UPDATE and
DELETE to your heart's content!

CONGRATULATIONS!



Practice Time!

- A. Rachel and Chandler (from our “Try This!” earlier) have been signed up to teach PROG8080 in the Winter 2020 session. Rachel is teaching Section 1 with a capacity of 47 students and Chandler is teaching Section 2 with a capacity of 34 students. Nobody is currently enrolled. Can you figure out how to INSERT the data into the dbo.CourseOffering table?
- B. We want to create a new room in the SIS dbo.Room table but all we know is the room number is “3G11”, the campus is Waterloo, the capacity is 48 the DVP is “Yes”, it doesn’t have a Robo Tel, it is a lab and it has a printer named “Greatest Printer Ever!”. How do we do this?
- C. In SIS, we want to insert every student who has a postal code beginning with “N2G” into the semester “2” of program code “1009” (dbo.StudentProgram table). The program status code will be “A”. How do we do it?
- D. We want to create a new table with data from the dbo.Person and dbo.Student table. This table would only show international student information and would include the following information: Student Number, Full Name (join the last and first), Personal Email, Student Local Phone Number
- E. We want to remove the table that we just created from the database. What’s the SQL to do it?



Summary

- Review Session Concepts
- This Week's Assignment
- Current Assignment Reminder
- Questions?



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

Fall 2019 - Week 6 – Thursday Lesson



Agenda

- In-Class Assignment
- Break
- Review of Last Week's Questions
- Midterm Review “Quiz”
- Q&A

Assignment #6 – In-Class Component

Rules:

- No talking or working with anyone else
- If you have any questions, ask me and nobody else
- Get the assignment from eConestoga
 - Follow all of the directions exactly regarding what to include
- When done, submit your in-class work through eConestoga
- If you’re done early bring me your laptop to confirm that you have uploaded your document to eConestoga and then please leave the room for a break and come back for the midterm review at 5:00pm



Assignment #6 – In-Class Component

Rules:

TIME REMAINING: 50 minutes

- No talking or working with anyone else
- If you have any questions, ask me and nobody else
- Get the assignment from eConestoga
 - Follow all of the directions exactly regarding what to include
- When done, submit your in-class work through eConestoga
- If you’re done early bring me your laptop to confirm that you have uploaded your document to eConestoga and then please leave the room for a break and come back for the midterm review at 5:00pm



Assignment #6 – In-Class Component

Rules:

TIME REMAINING: 40 minutes

- No talking or working with anyone else
- If you have any questions, ask me and nobody else
- Get the assignment from eConestoga
 - Follow all of the directions exactly regarding what to include
- When done, submit your in-class work through eConestoga
- If you’re done early bring me your laptop to confirm that you have uploaded your document to eConestoga and then please leave the room for a break and come back for the midterm review at 5:00pm



Assignment #6 – In-Class Component

Rules:

TIME REMAINING: 30 minutes

- No talking or working with anyone else
- If you have any questions, ask me and nobody else
- Get the assignment from eConestoga
 - Follow all of the directions exactly regarding what to include
- When done, submit your in-class work through eConestoga
- If you’re done early bring me your laptop to confirm that you have uploaded your document to eConestoga and then please leave the room for a break and come back for the midterm review at 5:00pm



Assignment #6 – In-Class Component

Rules:

TIME REMAINING: 20 minutes

- No talking or working with anyone else
- If you have any questions, ask me and nobody else
- Get the assignment from eConestoga
 - Follow all of the directions exactly regarding what to include
- When done, submit your in-class work through eConestoga
- If you’re done early bring me your laptop to confirm that you have uploaded your document to eConestoga and then please leave the room for a break and come back for the midterm review at 5:00pm



Assignment #6 – In-Class Component

Rules:

TIME REMAINING: 10 minutes

- No talking or working with anyone else
- If you have any questions, ask me and nobody else
- Get the assignment from eConestoga
 - Follow all of the directions exactly regarding what to include
- When done, submit your in-class work through eConestoga
- If you’re done early bring me your laptop to confirm that you have uploaded your document to eConestoga and then please leave the room for a break and come back for the midterm review at 5:00pm



Assignment #6 – In-Class Component

Rules:

TIME REMAINING: 5 minutes

- No talking or working with anyone else
- If you have any questions, ask me and nobody else
- Get the assignment from eConestoga
 - Follow all of the directions exactly regarding what to include
- When done, submit your in-class work through eConestoga
- If you’re done early bring me your laptop to confirm that you have uploaded your document to eConestoga and then please leave the room for a break and come back for the midterm review at 5:00pm



Assignment #6 – In-Class Component

Rules:

TIME REMAINING: 2 minutes

- No talking or working with anyone else
- If you have any questions, ask me and nobody else
- Get the assignment from eConestoga
 - Follow all of the directions exactly regarding what to include
- When done, submit your in-class work through eConestoga
- If you’re done early bring me your laptop to confirm that you have uploaded your document to eConestoga and then please leave the room for a break and come back for the midterm review at 5:00pm

Assignment #6 – In-Class Component

Rules:

TIME REMAINING: 1 minute

- No talking or working with anyone else
- If you have any questions, ask me and nobody else
- Get the assignment from eConestoga
 - Follow all of the directions exactly regarding what to include
- When done, submit your in-class work through eConestoga
- If you’re done early bring me your laptop to confirm that you have uploaded your document to eConestoga and then please leave the room for a break and come back for the midterm review at 5:00pm



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE®

ASSIGNMENT TIME IS UP

Make sure to upload your assignment to eConestoga
BREAK TIME!



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE®

Midterm Review

Topics Covered on the Midterm Exam Include:



Topics Covered on the Midterm Exam Include:

- Relational vs. Non-Relational DB
- OLTP, OLAP Databases
- Data Warehouses
- Various Data Types
- NULL values
- Precision, Scale & Length
- Table Keys (Candidate, Primary, Natural, Surrogate, Composite, Foreign)
- Cardinality
- Data/Relational Integrity
- T-SQL
 - Comments
 - SELECT Statement & Components
 - Coding Standards
 - Table Joins (Inner/Outer)
 - Aggregate Functions
 - GROUP BY / HAVING
 - Subqueries
 - UPDATE
 - DELETE / TRUNCATE
 - INSERT
 - Transactions



Last Week's Questions Review - I

- A. Rachel and Chandler (from our “Try This!” earlier) have been signed up to teach PROG8080 in the Winter 2020 session. Rachel is teaching Section 1 with a capacity of 47 students and Chandler is teaching Section 2 with a capacity of 34 students. Nobody is currently enrolled. Can you figure out how to **INSERT** the data into the dbo.CourseOffering table?

```
USE SIS
GO
INSERT INTO dbo.CourseOffering (sessionCode, courseNumber, sectionNumber,
employeeNumber, capacity, enrollment )
VALUES( 'w20', 'PROG8080', 1, 999998, 47, 0 )

INSERT INTO dbo.CourseOffering (sessionCode, courseNumber, sectionNumber,
employeeNumber, capacity, enrollment )
VALUES( 'w20', 'PROG8080', 2, 999999, 34, 0 )
```



Last Week's Questions Review - II

- B. We want to create a new room in the SIS dbo.Room table but all we know is the room number is “3G11”, the campus is Waterloo, the capacity is 48 the DVP is “Yes”, it doesn’t have a Robo Tel, it is a lab and it has a printer named “Greatest Printer Ever!”. How do we do this?

```
USE SIS
GO
INSERT INTO dbo.Room (number, campusCode, capacity, dvp, robotel, isLab, printer)
VALUES( '3G11', 'W', 48, 1, 0, 1, 'Greatest Printer Ever!')
```



Last Week's Questions Review - III

- C. In SIS, we want to insert every student who has a postal code beginning with “N2G” into the semester “2” of program code “1009” (dbo.StudentProgram table). The program status code will be “A”. How do we do it?

```
USE SIS
GO
INSERT INTO dbo.StudentProgram (studentNumber, programCode,
                                semester, programStatusCode)
SELECT s.number, '1009', 2, 'A'
      FROM dbo.Student AS s
     WHERE s.localPostalCode LIKE 'N2G%'
```



Last Week's Questions Review - IV

- D. We want to create a new table with data from the dbo.Person and dbo.Student table. This table would only show international student information and would include the following information: Student Number, Full Name (join the last and first), Personal Email, Student Local Phone Number

```
SELECT p.number,
       p.lastName + ', ' + p.firstName AS fullName,
       p.personalEmail,
       s.localPhone
  INTO dbo.internationalStudentInfo
   FROM dbo.Person AS p
 INNER JOIN dbo.Student AS s ON s.number = p.number
 WHERE s.isInternational = 1

SELECT * FROM dbo.internationalStudentInfo
```

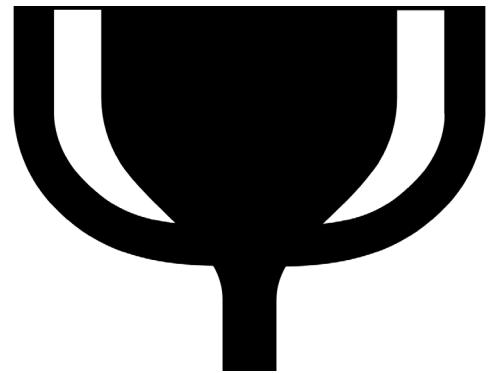


Last Week's Questions Review - V

- E. We want to remove the table that we just created from the database.
What's the SQL to do it?

```
DROP TABLE dbo.internationalStudentInfo
```

WHAT YOU DO HERE...
COUNTS OUT THERE.®



Quiz Time!

Questions?





CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

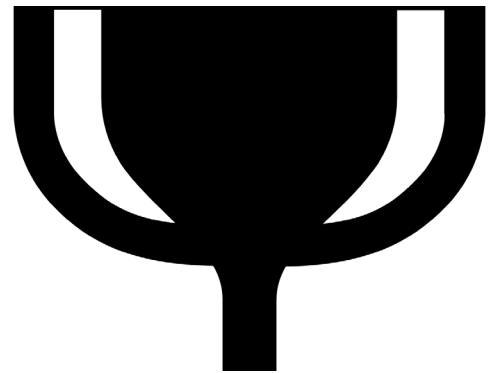
Fall 2019 - Week 7 – Friday Lesson (Mid-term Exam)



Agenda

- In-Class Assignment Make-up
- Review of Assignment #4
- Midterm Review “Quiz”
- Q&A
- Break
- Midterm Exam

WHAT YOU DO HERE...
COUNTS OUT THERE.®



Quiz Time!

Questions?

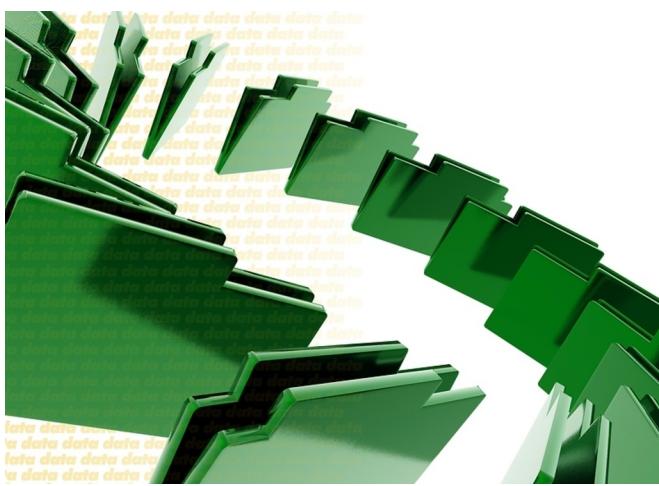




Mid-term Examination Time

Rules:

- No talking or working with anyone else or using other material
- If you have any questions, ask me and nobody else
- Access the mid-term on eConestoga
- Exam Notes:
 - The 60 questions are in random order per student
 - There is one question per screen and you cannot go back to previous ones
 - You have 50 minutes
- When you are done make sure to submit your results



WEEK #9

ASSIGNMENT

2% Individual Assignment

Database with a View

Mark Morell

Database Management – Fall 2019

Assignment Type:

- Individual – Prepare and submit your results independently

Date Due:

- Thursday, November 7th by the end of the day

Instructions:

- Please submit your assignment electronically through eConestoga.
- Assignments should be submitted as Microsoft Word files using the course coversheet format. You **MUST** include your query as text in the Word document as well as a **FULL screenshot** of your SSMS screen (screen capture the entire application screen including the title bar through the bottom of the window). Multiple screenshots may be required.
- If you are using external sources (images, text, etc.) you must reference them as part of your assignment and not copy them as-is.
- Best practice is to research your answers and then write the response to the question in your own words.
- Please include the question number with your responses.

Late Assignment Penalty:

Days Late	Penalty %
1	5
2	10
3	20
4	40
5	60
6	80
7	100

Assignment Questions

Question #	Question	Score
1	Suppose that you have a large number of individual updates (i.e. that involve multiple SQL statements) that you need to make to a database in a short period of time. Which of the three transactions modes (auto-commit, explicit and implicit) would be best to use and why?	2
2	Write a series of SQL statements that demonstrate the concept of a “dirty read”. Include comments in your SQL that describe why it demonstrates it.	2
3	<p>Using the SIS database, create a view named vStudentCourseList that produces the following information:</p> <ul style="list-style-type: none"> • Student Number • Student's Full Name • Course Name • Course Credits <p><i>Hint: You will need to include 5 tables in your view including dbo.Student, dbo.CourseStudent, dbo.CourseOffering, dbo.Course and dbo.Person to get the information required here</i></p>	4
4	Alter the vStudentCourseList created above to add the student's finalMark to the view and only include non-zero grades in the view.	2
5	<p>Using the view created above, write a query to retrieve the following information:</p> <ul style="list-style-type: none"> • Student Number • Student's Full Name • Student's Average of all Courses <p>Sort the information in the single query both by their average in descending order and then by student number in ascending order</p>	2
6	Using a Common Table Expression and <u>without</u> using any views, write a query to retrieve the exact same information as Question #5 above with the same columns and the same sort order. <i>Hint: Basically convert your view into a CTE and write a query around it.</i>	3
	Total	15



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

Fall 2019 - Week 9 – Monday Lesson



Agenda

- Daily Check-In / Administrivia
- Last Week's Questions Review
- Mid-Term Exam Review
- Transactions
- Summary



The Daily Dose

- Thumb-Ball Time!
- Check-In With Everyone
- Current Assignment Status
- Use www.menti.com to ask me anything
 - Code is on the board



Fun “Data” of the Day

Source: <https://www.livin3.com/50-cool-and-weird-fun-facts-that-you-should-know>

A sheep, a duck and a rooster were the first passengers in a hot air balloon

Mountain lions can whistle



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE®

Fun “Comic” of the Day

Source: <https://www.dilbert.com>

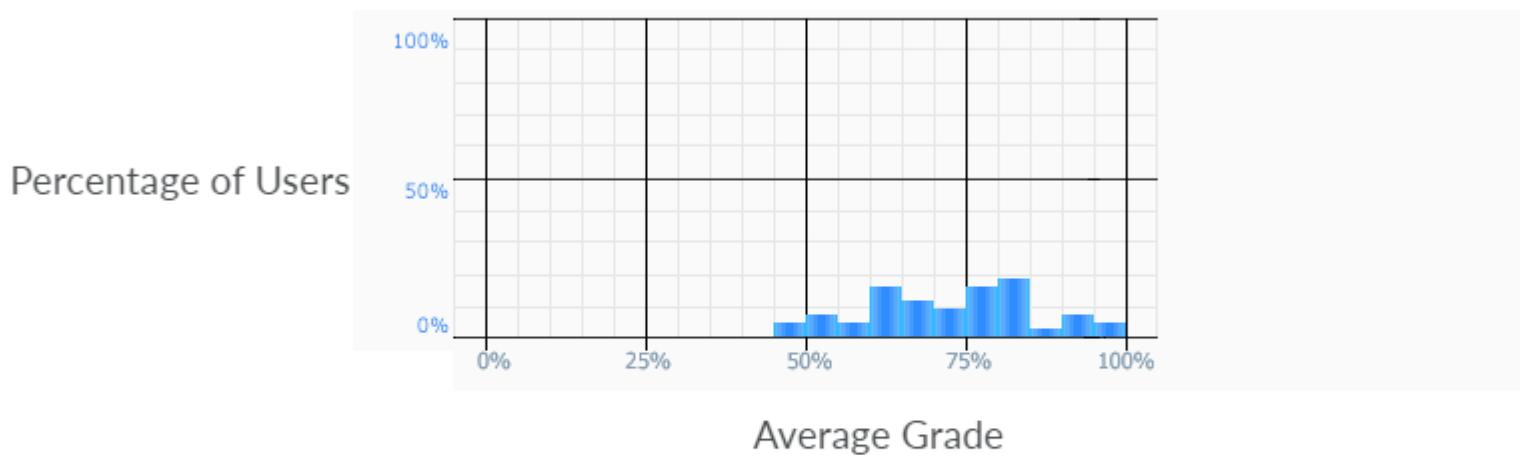




Recent Assignments and Mid-Term

- Assignment #5 Solutions
- Mid-Term Solutions

Score Distribution:



Class Average:



71.63 %



TRANSACTIONS (Review)

A transaction is a set of T-SQL statements that are executed together as a unit

- If successful, the data is *committed* and the changes made by the statements are permanently saved to the database
- User has the opportunity to *rollback* the changes so that nothing is applied to the database
- Basic syntax:

```
BEGIN { TRAN | TRANSACTION }
-- SQL statements here
{COMMIT | ROLLBACK} [ {TRAN | TRANSACTION} ]
```

Naming a Transaction

You can name a transaction so that you can work with it as a unit-of-work within a script. You declare it and then use it throughout your code as needed.

– Example:

```
USE SIS
GO
DECLARE @TransactionName varchar(20) = 'MyTransaction'

BEGIN TRAN @TransactionName

UPDATE dbo.Person SET...

ROLLBACK TRAN @TransactionName;
```



Database Transaction Properties

ACID is a set of properties to guarantee validity of the data in the event of problems (power outages, errors, etc.)

- Example: Financial transaction

A sequence of database operations that satisfies these ACID properties is a transaction





ACID

Atomicity

- The transaction is treated as a single *unit*
 - Succeeds completely or fails completely

Consistency

- Data in a consistent state and the start and end of transaction



More ACID

Isolation

- The middle state(s) of a transaction are invisible to other transactions
 - Concurrent (running at the same time) transactions appear to be run in as a series

Durability

- Guarantee that once complete and committed, it stays committed even in the event of a system failure
 - Not in “memory” and no reversal of the changes



Database Transactions Modes

There are 3 types of transaction “modes”:

- Autocommit Transactions
 - The “default” mode that we’ve used mostly to-date
 - SQL statements automatically committed and rolled back
 - Best for singleton changes that aren’t related to other database changes
 - With a large volume of statements, this mode commits after every one and so can be database intensive

Database Transactions Modes - Explicit

- Explicit Transactions
 - Transactions that start with BEGIN TRANSACTION (or BEGIN TRAN) and are completed with a COMMIT or undone with a ROLLBACK statement
 - Allows you to specifically create a “unit of work” for database changes
 - Also useful (performance-wise) for a large set of database updates



Database Transactions Modes - Implicit

- Implicit Transactions
 - Enabled with the following syntax:

```
SET IMPLICIT_TRANSACTIONS ON
```

- Puts you in a mode where whenever data is modified (INSERT, UPDATE, DELETE) a transaction is automatically started
- The data is not committed until a COMMIT is provided
- The most confusing transaction mode
 - Need to remember to COMMIT transactions and harder to define the “unit of work”



Implicit Transaction Try This

Q: Using SIS, set the transaction mode to implicit and go ahead update and delete data in the dbo.CourseOffering table. Query the data after you do the operation but don't commit the data. Instead, after a few SQL statements, use ROLLBACK to undo your changes. Query the data to see if it's in the database or not.

```
SET IMPLICIT_TRANSACTIONS ON
```



Database Transaction Terms - I

Dirty Read

- A read (SELECT) from the database of data that has not yet been committed
 - e.g. The SELECT statements being executed in the implicit transaction before being COMMITTED were “dirty reads”



Database Transaction Terms - II

Non-Repeatable Read

- Occurs during the course of a transaction where a SELECT retrieves the same row twice and the values differ
 - Example:
 - User A runs the same query twice
 - At the same time, User B runs a transaction and COMMITS an UPDATE of data
 - The row that User A queried has a different result the second time



Database Transaction Terms - III

Phantom Read

- Occurs during the course of a transaction two identical queries are executed and the set of rows returned by the second query is different than the first
 - Example:
 - User A runs the same query twice
 - At the same time, User B runs a transaction and COMMITS a change with either an INSERT or DELETE of data
 - The rows that User A queried are different the second time



Transaction Isolation Levels

Q: How do you manage these different sets of results so that the data queried makes sense to the user?

A: Transaction Isolation Levels

The DBMS acquires *locks* on data that may result in loss of concurrency but allows for the data to be more meaningful (i.e. removing phantom rows, non-repeating read situations)



SQLServer Isolation Levels

Five Levels Available

– Syntax:

```
SET TRANSACTION ISOLATION LEVEL
  { READ UNCOMMITTED
  | READ COMMITTED
  | REPEATABLE READ
  | SNAPSHOT
  | SERIALIZABLE
  }
```

Default Level: READ COMMITTED



Isolation Level Descriptions

READ UNCOMMITTED

- SQL statements can read rows that have been modified by other transactions but not yet committed to the database

READ COMMITTED

- SQL statements cannot read rows that have been modified by other transactions but not yet committed to the database
- Prevents “dirty reads”



More Isolation Level Descriptions

REPEATABLE READ

- Like READ COMMITTED but also no other transactions can modify data that has been read by the current transaction until the current transaction has completed

SNAPSHOT

- Data read a transaction will be the “transactional consistent” version of the data that existed at the start of the transaction. Data changes made by other transactions after the start of the current transaction are not visible to statements executing in the current transaction



Even More Isolation Level Descriptions

SERIALIZABLE

- The most strict of the isolation levels:
 - Statements cannot read data that has been modified by not yet committed in other transactions
 - No other transaction can modify data that has been read by the current transaction until it completes
 - Other transactions cannot insert new rows with key values that would fall in the range of keys read by any statements in the current transaction until it completes

Concurrent Access & Performance

Notes:

- Transaction levels can be switched during the transaction with the exception of changing *to* the SNAPSHOT isolation level
- It's a balancing act to provide “good” data but also sharing access to the data so that users aren't blocked
 - It impacts performance if transactions have to wait for others to finish
 - **Deadlock** occurs when two or more sessions have some data locked and each requires a lock on the other's data
- Important to choose the isolation level needed in your situation



The Transaction Log – Part I

Q: What is the transaction log used for?

A: It is used by SQL Server to capture all transaction information. Key points include:

- Record of all changes made to the database while actual data is stored in a separate file
- Contains enough information to undo changes made in a transaction
- Recover incomplete transactions as SQL Server starts
- Restoration of data up to a point of failure (disaster recovery)
- Supports transactional replication and database mirroring



The Transaction Log – Part II

Q: How can you look at the transaction log?

A: Use the fn_dblog function

```
USE dbname
GO
SELECT * FROM fn_dblog(NULL, NULL )
```

- It takes a starting and ending Log Sequence Number as parameters
- Pass NULL for both if you want to see all of the data
- Cryptically shows all transaction information including locks, etc.
- Hard to make sense of – there are tools to help make this readable



Transaction Log - Try This!

In SSMS, have a look at the transaction log for one of the databases (i.e. SIS). Then try the following:

- i. Count how many entries are in the log
- ii. Perform a transaction of some sort (SELECT, UPDATE, etc.)
- iii. Now see how many entries are in the log
- iv. Repeat steps ii. and iii. to see how the log grows over time



Summary

- Review Session Concepts
- Questions?



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

Fall 2019 - Week 9 – Thursday Lesson



Agenda

- Daily Check-In / Administrivia
- Terminology/Topic Review
- Database Objects
 - Views
 - Common Table Expressions (CTEs)
- Practice!
- Summary



The Daily Dose

- Thumb-Ball Time!
- Check-In With Everyone
- Current Assignment Status
- Mid-Term Information
- Use www.menti.com to ask me anything
 - Code is on the board



Fun “Data” of the Day

Source: <http://www.halloween-website.com/trivia.htm>

Halloween was brought to North America by immigrants from Europe who would celebrate the harvest around a bonfire, share ghost stories, sing, dance and tell fortunes

The Ouija Board ended up outselling the game of Monopoly in its first full year at Salem. Over two million copies of the Ouija Board were shipped

Recent Terminology & Topics

- Transactions
 - ACID: Atomicity, Consistency, Isolation, Durability
 - Modes: Auto-commit, Explicit & Implicit
 - Dirty Read
 - Non-Repeatable Read
 - Phantom Read
- SQL Server Isolation Levels
- Transaction Log



Other Database Objects

So far, we've been dealing mostly with database tables but there are other database objects:

- Views
- Common Table Expressions (CTEs)
- Stored Procedures / Functions
- Triggers
- Constraints
- Indices



Database Views

In SQL, a view is a *virtual table* based on the results of a SQL query

- It contains rows and columns just like tables
- The fields in the view are fields from one or more tables in the database
- Data in the view *can* be updated but there are rules for what can be updated and it's not considered good practice to update the data in this manner

Why Use a View?

There are a number of reasons to use a view:

- To simplify or customize the user's understanding of the database
 - For example, create a view of often-used data that joins between multiple tables so that developers/users don't have to write the complex query every time they need the data
- Security Reasons
 - You may want to protect certain table data from being seen and by only giving user access to a view you can “hide” that sensitive data

Why Use a View? Part II

There are a number of reasons to use a view:

- They can be used to support legacy code
 - For example, if you need to refactor a table that would break application code you could create a view with the same name as the table that has the same columns and then make whatever changes you need to make to the table



View Creation Syntax

```
CREATE VIEW [schema_name.]view_name
    [ WITH { ENCRYPTION | SCHEMABINDING } ]
AS
    SELECT expressions
        FROM tables
    [WHERE conditions]
    [ WITH CHECK OPTION ]
```

Notes:

- Name the view prefixed with a “v” or “view” so that it stands out
- View Options:
 - **ENCRYPTION** – encrypt the view source so that the contents of the actual query can’t be seen by users
 - **SCHEMABINDING** – ensure underlying table definition cannot be modified
 - **CHECK OPTION** – enforce update/insert on view to be constrained by view criteria



View Creation / Use Example

```
USE SIS
GO
CREATE VIEW dbo.vInternationalStudent AS
    SELECT p.lastName + ' , ' + p.firstName AS fullName,
           p.personalEmail,
           p.localPhone,
           p.localStreet,
           p.localCity,
           p.localProvinceCode,
           p.localPostalCode
      FROM dbo.Person AS p
     INNER JOIN dbo.Student AS s ON s.number = p.number
    WHERE s.isInternational = 1
```

```
SELECT * FROM vInternationalStudent
```

```
SELECT fullName, personalEmail
      FROM vInternationalStudent
     WHERE fullName LIKE '%w%'
```



View Commands

To see the contents of a view use the command:

```
sp_helptext '[schema_name.]view_name'
```

To change the content of a view use the ALTER VIEW command:

```
ALTER VIEW [schema_name.]view_name
[ WITH { ENCRYPTION | SCHEMABINDING } ]
AS
SELECT expressions
      FROM tables
[WHERE conditions]
[ WITH CHECK OPTION ]
```

To delete the view, use the DROP VIEW command:

```
DROP VIEW [schema_name.]view_name
```

Views – Try This!

- i. In SIS, create a brand new view named *vDoonEmployee* that gets employees whose “campusCode” is “D”. Create the view as an ENCRYPTED view with the following columns:
 - Full Name (Last, First Name)
 - College Email
 - Personal Email
- ii. Write SQL that uses the view to select all Doon employees who have “net” in their personal email
- iii. Look at the source of your view. What do you see?
- iv. Update your view to add the employee phone number
- v. Remove your view from the database

```
CREATE VIEW [schema_name.]view_name  
AS  
    SELECT expressions  
        FROM tables  
    [WHERE conditions]
```



Views – Try This! Solution

```
USE SIS
GO

-- Create the view:
CREATE VIEW dbo.vDoonEmployee
WITH ENCRYPTION
AS
    SELECT p.lastName + ', ' + p.firstName AS fullName,
           p.collegeEmail,
           p.personalEmail
      FROM dbo.Person AS p
     INNER JOIN dbo.Employee AS e ON e.number = p.number
    WHERE e.campusCode = 'D'

-- Get employee information with "net" in their email:
SELECT * FROM dbo.vDoonEmployee WHERE personalEmail LIKE '%net%'
```



Views – Try This! Solution Part II

```
-- Look at the view:  
sp_helptext 'dbo.vDoonEmployee'  
  
-- Modify the view to add phone number:  
ALTER VIEW dbo.vDoonEmployee  
WITH ENCRYPTION  
AS  
    SELECT p.lastName + ', ' + p.firstName AS fullName,  
          p.collegeEmail,  
          p.personalEmail,  
          p.mainPhone  
     FROM dbo.Person AS p  
INNER JOIN dbo.Employee AS e ON e.number = p.number  
   WHERE e.campusCode = 'D'  
  
-- Remove the view:  
DROP VIEW dbo.vDoonEmployee
```

Common Table Expressions (CTEs)

In SQL Server, a CTE is used to create a temporary table that exists for the life of a query.

- It is similar to a view but is not maintained post-query
- It can be used with SELECT, INSERT, DELETE or UPDATE statements
- It can also be used with the creation of a VIEW
- You can use multiple CTEs in a query (separate with commas)



Why Use a CTE?

There are a number of reasons to use a CTE:

- Simplification of complicated queries
- Use as a substitute for a view when there's no good reason to create a view
- Recursion – allows coding a query that references itself which can be very handy (e.g. organizational chart type use)



CTE Creation Syntax

```
WITH CTE_name (expression1, expression2, ...)
AS
-- Define the CTE query:
( SELECT expression1, expression1, ...
      FROM table
     WHERE condition(s)
)
-- Define the query referencing the CTE
SELECT expression(s)
      FROM CTE_name ...
```

Notes:

- The CTE must be followed by a single SELECT, INSERT, UPDATE or DELETE statement that references some or all of the CTE columns
- You cannot use more than one WITH clause in a CTE so if you had a subquery in a CTE it cannot define another CTE within it
- You cannot use ORDER BY within a CTE



CTE Example

```
USE Adventureworks2017
GO
-- Define the CTE expression name and column list.
WITH Sales_CTE (SalesPersonID, SalesorderID, SalesYear)
AS
-- Define the CTE query.
(
    SELECT SalesPersonID, SalesOrderID, YEAR(OrderDate) AS SalesYear
        FROM Sales.SalesOrderHeader
        WHERE SalesPersonID IS NOT NULL
)
-- Define the outer query referencing the CTE name.
SELECT SalesPersonID, COUNT(SalesOrderID) AS Totalsales, SalesYear
    FROM Sales_CTE
    GROUP BY SalesYear, SalesPersonID
    ORDER BY SalesPersonID, SalesYear;
```

CTEs – Try This!

- i. Using AdventureWorks2017, create a query using a CTE that retrieves the average number of sales order (from the Sales.SalesOrderHeader table) for all years for the sales representatives.
- ii. In SIS, create a query using a CTE to traverse the dbo.Employee table to get the following information:
 - Manager's ID (reportsTo column)
 - Employee's ID (number column)
 - Employee Level (number of people above them reporting-wise. 0 for the top-level)

```
WITH CTE_name (expression1, expression2, ...)  
AS  
( SELECT expression1, expression1, ...  
    FROM table  
    WHERE condition(s)  
)  
SELECT expression(s)  
FROM CTE_name ...
```



CTE Try This!

```
USE Adventureworks2017
GO
WITH Sales_CTE (SalesPersonID, Numberoforders)
AS
(
    SELECT SalesPersonID, COUNT(*)
        FROM Sales.SalesOrderHeader
       WHERE SalesPersonID IS NOT NULL
         GROUP BY SalesPersonID
)
SELECT AVG(Numberoforders) AS averagesalesPerPerson
      FROM Sales_CTE;
```



Recursive CTE Try This!

```
USE SIS
GO
WITH DirectReports(ManagerID, EmployeeID, EmployeeLevel) AS
(
    SELECT reportsTo, number, 0 AS EmployeeLevel
    FROM dbo.Employee
    WHERE reportsTo IS NULL
    UNION ALL
    SELECT e.reportsTo, e.number, EmployeeLevel + 1
    FROM dbo.Employee AS e
        INNER JOIN DirectReports AS d
        ON e.reportsTo = d.EmployeeID
)
SELECT ManagerID, EmployeeID, EmployeeLevel
FROM DirectReports
ORDER BY ManagerID, EmployeeID
```



Practice Time!

- A. Using AdventureWorks2017, create a new view named vVendorPOTotalOrdered that select the following data from the Purchasing.PurchaseOrderHeader and Purchasing.PurchaseOrderDetail tables:
 - Vendor ID
 - Purchase Order ID
 - Sum of the Ordered Quantity by the Vendor ID and Purchase Order ID
- B. Retrieve all of the data from your new view above where the total ordered quantity is more than 500. Order the results by ordered quantity in descending order.
- C. Modify your view to include the date part (i.e. no time) of the OrderDate from the header table
- D. Run the command to show the contents of your view
- E. Delete the new view
- F. Using SIS, create a query using a CTE that calculates the average enrollment across all courses in the dbo.CourseOffering table where the CTE retrieves the courseNumber and total enrollment for that course.



Summary

- Review Session Concepts
- This Week's Assignment
- Questions?



CONESTOGA
Connect Life and Learning

WHAT YOU DO HERE...
COUNTS OUT THERE.®

Database Management (PROG8080)

Fall 2019 - Week 10 – Monday Lesson



Agenda

- Daily Check-In / Administrivia
- Terminology/Topic Review
- Database Objects
 - Stored Procedures
 - Functions
- Practice!
- Summary



The Daily Dose

- Thumb-Ball Time!
- Check-In With Everyone
- Current Assignment Status
- Use www.menti.com to ask me anything
 - Code is on the board



Fun “Data” of the Day

Source: <https://www.livin3.com/50-cool-and-weird-fun-facts-that-you-should-know>

The largest recorded snowflake was in Keogh, MT during year 1887, and was 15 inches wide

10% of the World's population is left handed



Recent Terminology & Topics

- Views
- Common Table Expressions (CTEs)
 - Recursion



Last Week's Questions Review - I

- A. Using AdventureWorks2017, create a new view named vVendorPOTotalOrdered that select the following data from the Purchasing.PurchaseOrderHeader and Purchasing.PurchaseOrderDetail tables:
- Vendor ID
 - Purchase Order ID
 - Sum of the Ordered Quantity by the Vendor ID and Purchase Order ID

```
CREATE VIEW Purchasing.vVendorPOTotalordered
AS
SELECT poh.VendorID,
       poh.PurchaseOrderID,
       SUM( pod.OrderQty ) AS sumorderQty
  FROM Purchasing.PurchaseOrderHeader AS poh
INNER JOIN Purchasing.PurchaseOrderDetail AS pod
         ON pod.PurchaseOrderID = poh.PurchaseOrderID
 GROUP BY poh.vendorID, poh.PurchaseOrderID
```



Last Week's Questions Review - II

- B. Retrieve all of the data from your new view above where the total ordered quantity is more than 500. Order the results by ordered quantity in descending order.

```
SELECT *
  FROM Purchasing.vVendorPOTotalOrdered
 WHERE sumOrderQty > 500
 ORDER BY sumOrderQty DESC
```



Last Week's Questions Review - III

- C. Modify your view to include the date part (i.e. no time) of the OrderDate from the header table

```
ALTER VIEW Purchasing.vVendorPOTotalOrdered
AS
SELECT poh.VendorID,
       poh.PurchaseOrderID,
       CONVERT( DATE, poh.orderDate ) AS orderDate,
       SUM( pod.OrderQty ) AS sumOrderQty
  FROM Purchasing.PurchaseOrderHeader AS poh
INNER JOIN Purchasing.PurchaseOrderDetail AS pod
         ON pod.PurchaseOrderID = poh.PurchaseOrderID
 GROUP BY poh.VendorID, poh.PurchaseOrderID, CONVERT( DATE, poh.OrderDate )
```



Last Week's Questions Review - IV

- D. Run the command to show the contents of your view

```
sp_helptext 'Purchasing.vVendorPOTotalOrdered'
```



Last Week's Questions Review - V

- E. Delete the new view.

```
DROP VIEW Purchasing.vVendorPOTotalOrdered
```



Last Week's Questions Review - VI

- E. Using SIS, create a query using a CTE that calculates the average enrollment across all courses in the dbo.CourseOffering table where the CTE retrieves the courseNumber and total enrollment for that course.

```
USE SIS
GO
WITH courseEnrollment (courseNumber, totalEnrollment)
AS
(
    SELECT courseNumber, SUM( enrollment ) courseTotalEnrollment
        FROM dbo.CourseOffering
       GROUP BY courseNumber
)
SELECT AVG(totalEnrollment) AS averageCourseEnrollment
      FROM courseEnrollment
```



Other Database Objects

After tables there are other database objects including:

- ✓ – Views
- ✓ – Common Table Expressions (CTEs)
- Stored Procedures
- Functions
- Triggers
- Constraints
- Indices



Stored Procedures

In SQL, a stored procedure is a group of one or more T-SQL statements in a logical unit that performs some operation that may be executed within the database.

- It persists within the database like other objects
- It can have input and/or output parameters
- It can declare and use variables within the stored procedure
- It can utilize flow control statements (IF/ELSE, WHILE, etc.)
- It provides exception handling

Why Use a Stored Procedure?

There are a number of reasons to use a stored procedure:

- Allow for Reusable Modular Programming
 - All logic is encapsulated in the stored procedure and can be reused
- Allows for Conditional Logic
- Allow for Faster Execution / Reduction in Network Traffic
 - Logic run (and cached) entirely on the server-side and optimized there
- Security Reasons
 - You may want to protect certain algorithms, table data look-up from being seen and so you can “hide” that sensitive data or process

Stored Procedure Disadvantages

They do have some drawbacks:

- Limited Coding Functionality
 - Logic not as robust as app code
- Portability
 - Complex Stored Procedures may not port to upgraded versions easily
- Cost
 - May require a separate SQL developer to maintain



Simple Stored Procedure Creation Syntax

```
CREATE PROCEDURE [schema_name.]storedprocedure_name
AS
BEGIN
    SELECT expressions
        FROM tables
        [WHERE conditions];
END;
```

Notes:

- Name the stored proc prefixed with a “sp” or “sp_” so that it stands out
- Statements in the stored procedure must end with a semi-colon (;)
- Logic is wrapped by BEGIN and END statements



Simple Stored Procedure Execution Syntax

```
EXECUTE [schema_name.] storedprocedure_name;
```

or

```
EXEC [schema_name.] storedprocedure_name;
```



Stored Procedure Creation / Use Example

```
CREATE PROCEDURE dbo.sp_internationalStudent
AS
BEGIN
    SELECT p.lastName + ' , ' + p.firstName AS fullName,
           p.personalEmail,
           p.localPhone,
           p.localStreet,
           p.localCity,
           p.localProvinceCode,
           p.localPostalCode
      FROM dbo.Person AS p
     INNER JOIN dbo.Student AS s ON s.number = p.number
     WHERE s.isInternational = 1;
END;
```

```
EXECUTE dbo.sp_internationalStudent;
```



Stored Procedure Commands

To see the contents of a stored procedure use the command:

```
sp_helptext '[schema_name.]storedprocedure_name'
```

To change the content of a stored proc use ALTER PROCEDURE:

```
ALTER PROCEDURE [schema_name.]storedprocedure_name
AS
BEGIN
    SELECT expressions
        FROM tables
    [WHERE conditions];
END;
```

To delete the stored procedure, use DROP PROCEDURE:

```
DROP PROCEDURE [schema_name.]storedprocedure_name
```

Stored Procedures – Try This!

- i. In SIS, create a brand new stored procedure named *sp_doonEmployee* that gets employees whose “campusCode” is “D”. Create the procedure to return the following information:
 - Full Name (Last, First Name)
 - College Email
 - Personal Email
- ii. Write SQL that uses the proc
- iii. Update your stored proc to add the employee phone number
- iv. Remove your stored proc from the database

```
CREATE PROCEDURE  
    [schema_name.] storedprocedure_name  
AS  
BEGIN  
    SELECT ...;  
END;
```



Stored Procedures – Try This! Solution

```
USE SIS
GO

-- Create the proc:
CREATE PROCEDURE dbo.sp_doonEmployee
AS
BEGIN
    SELECT p.lastName + ', ' + p.firstName AS fullName,
           p.collegeEmail,
           p.personalEmail
      FROM dbo.Person AS p
     INNER JOIN dbo.Employee AS e ON e.number = p.number
     WHERE e.campusCode = 'D';
END;

-- Execute the procedure:
EXECUTE dbo.sp_doonEmployee;
```



Stored Procs – Try This! Solution Part II

```
-- Modify the procedure to add phone number:  
ALTER PROCEDURE dbo.sp_doonEmployee  
AS  
BEGIN  
    SELECT p.lastName + ', ' + p.firstName AS fullName,  
          p.collegeEmail,  
          p.personalEmail,  
          p.mainPhone  
    FROM dbo.Person AS p  
    INNER JOIN dbo.Employee AS e ON e.number = p.number  
    WHERE e.campusCode = 'D';  
END;  
  
-- Remove the proc:  
DROP PROCEDURE dbo.sp_doonEmployee;
```



Stored Proc Creation With Parameters Syntax

```
CREATE PROCEDURE [schema_name.]storedprocedure_name
    @parameter1 AS datatype,
    @parameter2 AS datatype, ...
AS
BEGIN
    SELECT expressions
        FROM tables
        WHERE conditions = @parameter1...
END;
```

Notes:

- Use the @ before the parameter name to identify it as a parameter
- Make your parameter name meaningful
- Reference it in your procedure logic with the @ symbol before the name



Stored Procedure Creation / Use Example 2

```
CREATE PROCEDURE dbo.sp_internationalStudent
    @lastName AS nvarchar(50)
AS
BEGIN
    SELECT p.lastName + ' , ' + p.firstName AS fullName,
        p.personalEmail,
        s.localPhone,
        s.localStreet,
        s.localCity,
        s.localProvinceCode,
        s.localPostalCode
    FROM dbo.Person AS p
    INNER JOIN dbo.Student AS s ON s.number = p.number
    WHERE s.isInternational = 1 AND
        p.lastName = @lastName;
END;
```

```
EXECUTE dbo.sp_internationalStudent N'WANG';
```



Stored Procedure Creation / Use Example 3

```
CREATE PROCEDURE dbo.sp_internationalStudent
    @lastNameLike AS nvarchar(50) = N'D%' ,
    @firstNameLike AS nvarchar(50) = N'%'
AS
BEGIN
    SELECT p.lastName + ' , ' + p.firstName AS fullName,
        p.personalEmail,
        s.localPhone,
        s.localStreet,
        s.localCity,
        s.localProvinceCode,
        s.localPostalCode
    FROM dbo.Person AS p
    INNER JOIN dbo.Student AS s ON s.number = p.number
    WHERE s.isInternational = 1 AND
        p.lastName LIKE @lastNameLike AND
        p.firstName LIKE @firstNameLike
END;
```



Stored Procedure Creation / Use Example 3b

```
EXECUTE dbo.sp_internationalStudent;  
  
EXECUTE dbo.sp_internationalStudent N'WANG', N'M%' ;  
  
EXECUTE dbo.sp_internationalStudent @firstNameLike N'N%' ;
```



Stored Procedure Variables

Variables are objects used to hold a single value of a certain type

- Used for loop counters, values to be tested, store values to be returned

Use DECLARE statement to create:

```
DECLARE @variableName AS datatype;
```

To assign a value to a variable use SET:

```
SET @variableName = value;
```



Stored Procedure Variable Use

```
CREATE PROCEDURE dbo.sp_internationalStudent
    @lastName AS nvarchar(50)
AS
BEGIN
    DECLARE @provinceCode nchar(2);

    SET @provinceCode = N'ON';

    SELECT p.lastName + ' ' + p.firstName AS fullName,
           p.personalEmail,
           s.localPhone,
           s.localStreet,
           s.localCity
      FROM dbo.Person AS p
     INNER JOIN dbo.Student AS s ON s.number = p.number
     WHERE s.isInternational = 1 AND
           p.lastName = @lastName AND
           s.localProvinceCode = @provinceCode;

END;
```



Stored Procedure Output Parameters

Stored Procedures can return output values

To use, simply specify OUTPUT after the parameter datatype

Then declare a variable before calling the stored procedure and handle the results



Stored Procedure Output Parameter Use

```
CREATE PROCEDURE dbo.sp_internationalStudent
    @lastName AS nvarchar(50),
    @personCount AS int OUTPUT
AS
BEGIN
    SELECT p.lastName + ' , ' + p.firstName AS fullName,
        p.personalEmail,
        s.localPhone,
        s.localStreet,
        s.localCity
    FROM dbo.Person AS p
    INNER JOIN dbo.Student AS s ON s.number = p.number
    WHERE s.isInternational = 1 AND
        p.lastName = @lastName;

    SELECT @personCount = @@ROWCOUNT;
END;
```

```
DECLARE @count int;

EXECUTE dbo.sp_internationalStudent
    @lastName = N'WANG',
    @personCount = @count OUTPUT;

PRINT 'Total Number of rows'
PRINT @count

-- Or could do this:
SELECT @count AS numberofPersonRecords;
```



Stored Procedure Flow Control

Like many programming languages, stored procedures can use various mechanisms of controlling the path through their logic including:

- BEGIN...END – create a block of T-SQL that executes together
- IF ELSE – a statement block based on a condition
- WHILE – repeatedly execute commands based on a condition
- BREAK – for exiting loops immediately
- CONTINUE – for skipping the current iteration of a loop
- Exception Handling
 - Provides TRY CATCH, RAISERROR and THROW functionality



Cursors

Cursors

- Allows processing of the rows of a result set one at a time
- Used as a mechanism within a stored procedure to investigate and process each row of a query one-by-one

```
DECLARE cursor_name CURSOR
      FOR select_statement;  
  
OPEN cursor_name  
  
WHILE @@FETCH_STATUS = 0
BEGIN
    FETCH NEXT FROM cursor_name INTO variable_list;
END  
  
CLOSE cursor_name;  
  
DEALLOCATE cursor_name;
```



Stored Procedure NOCOUNT

Use the following command to turn off SQL Server messages that are sent back to the client after any SELECT, INSERT, UPDATE, DELETE statements are executed in the stored procedure

```
SET NOCOUNT ON;
```

Put the code as the first statement in the body of the procedure after the AS keyword.

User-Defined Scalar Functions

Similar to a Stored Procedure but return a single value

- Created by users to simplify code by providing an easy way to determine the result of a complex calculation
- Same constructs as Stored Procedures available for use
- CAN be called within a stored procedure or any query
 - A stored procedure can't be called from within a function



Function Creation Syntax

```
CREATE FUNCTION [schema_name.] function_name
    (parameter_list)
RETURNS datatype AS
BEGIN
    statements
    RETURN value
END;
```

Notes:

- Name the stored proc prefixed with a “fn”, “fn_” or “ufn” (user function) so that it stands out
- Logic is wrapped by BEGIN and END statements



Function Creation / Use Example

```
CREATE FUNCTION sales.ufnNetSale(
    @quantity    INT,
    @list_price DEC(10,2),
    @discount    DEC(4,2)
)
RETURNS DEC(10,2) AS
BEGIN
    RETURN @quantity * @list_price * (1 - @discount);
END;
```

```
SELECT sales.ufnNetSale( 10, 12.50, 0.1 ) AS netsale;
```



Function Creation / Use Example II

```
CREATE FUNCTION Sales.ufnNumOrderDetail(
    @salesorderId    INT
)
RETURNS INT AS
BEGIN
    DECLARE @count INT;

    SELECT @count = COUNT( * )
        FROM Sales.SalesorderDetail
        WHERE SalesorderID = @salesorderId;

    RETURN @count;
END;
```

```
SELECT Sales.ufnNumOrderDetail( 43659 ) AS numOrderDetail
```



Function Creation / Use Example III

```
CREATE FUNCTION Sales.ufnProductorders(
    @ProductId    INT
)
RETURNS TABLE AS
RETURN
    SELECT SalesorderId, SalesOrderDetailId, orderQty
        FROM Sales.SalesOrderDetail
    WHERE ProductID = @ProductId;
```

```
SELECT * FROM Sales.ufnProductOrders( 716 )
```



Function Commands

To see the contents of a function use the command:

```
sp_helptext '[schema_name.]function_name'
```

To change the content of a function use ALTER FUNCTION:

```
ALTER FUNCTION [schema_name.]function_name  
    (parameter_list)  
RETURNS datatype AS  
BEGIN  
    statements  
    RETURN value  
END;
```

To delete the function, use DROP FUNCTION:

```
DROP FUNCTION [schema_name.]function_name
```



Scalar Function – Try This!

- i. In AdventureWorks2017, create a brand new function named ufnPersonFullName that takes an input parameter of the person's ID and builds a full name out of their Title, First Name, Middle Name and Last Name.
- ii. Write SQL that calls the function
- iii. Update the function to include Suffix and re-rerun your part ii.
- iv. Remove the function from the database

```
CREATE FUNCTION [schema_name.] function_name
    (parameter_list)
RETURNS datatype AS
BEGIN
    statements
    RETURN value
END;
```



Scalar Functions – Try This! Solution I

```
-- Create the function:  
CREATE FUNCTION Person.ufnPersonFullName(@personId INT)  
RETURNS nvarchar(158) AS  
BEGIN  
    DECLARE @fullName nvarchar(158);  
  
    SELECT @fullName = CONCAT( Title, ' ', FirstName, ' ', MiddleName, ' ', LastName )  
    FROM Person.Person  
    WHERE BusinessEntityID = @personId;  
  
    RETURN @fullName;  
END;  
  
-- Execute the SQL:  
SELECT Person.ufnPersonFullName( BusinessEntityID ) AS fullName,  
       Demographics  
  FROM Person.Person
```



Scalar Functions – Try This! Solution II

```
-- Alter the function:  
ALTER FUNCTION Person.ufnPersonFullName(@personId INT)  
RETURNS nvarchar(158) AS  
BEGIN  
    DECLARE @fullName nvarchar(158);  
  
        SELECT @fullName = CONCAT( Title, ' ', FirstName, ' ', MiddleName, ' ', LastName,  
        ' ', Suffix )  
        FROM Person.Person  
        WHERE BusinessEntityID = @personId;  
  
    RETURN @fullName;  
END;
```



Scalar Functions – Try This! Solution III

```
-- Execute the SQL:  
SELECT Person.ufnPersonFullName( BusinessEntityID ) AS fullName,  
       Demographics  
  FROM Person.Person  
 WHERE Suffix IS NOT NULL  
  
-- Finally drop the function:  
DROP FUNCTION Person.ufnPersonFullName
```



Practice Time!

A. “FREE-FOR-ALL!” – Stored Procedure

- Identify a need or two within the SIS, AdventureWorks2017 or Costume database as a class and come up with a stored procedure that will fill that need. Try to individually work out the details.

B. “FREE-FOR-ALL!” – Function

- Identify a need or two within the SIS, AdventureWorks2017 or Costume database as a class and come up with a user scalar function that will fill that need. Try to individually work out the details.



Summary

- Review Session Concepts
- This Week's Assignment
- Questions?