Database Management
Single Table SELECT

By: Meyer Tanuan (2022), Glenn Paulley (2015), John Mckay (2011)

SQL Coding Standards

- See **Standards\SQL Coding Standards.pdf** in eConestoga
- The primary goal of standards is readability
- Styles of writing SQL are as abundant as styles of writing other software artifacts
- However, you should follow these rules:
  - Major clauses should always begin on a new line
  - SQL language keywords should appear in UPPER CASE
  - Note that SQL is a case-agnostic language
  - Indent subqueries and derived tables in a FROM clause so that the major clauses of each line up together as a unit

CONESTOGA
Connect Life and Learning

Naming .sql files

- To practice for *Exercise 1* create a file named like this:

  `ex0.sql`

- For assignments, create a file named like this:

  `A0.sql`

- We'll add comments and SQL statements to this file as if it is an exercise or assignment

To show line numbers: Tools -> Options… -> Text Editor -> Transact-SQL -> General: check **Line numbers**

Comments in SQL

- Enter two dashes at the start of a line to mark a comment

```
-- This is a comment
```

- Use /* and a matching */ to mark a multi-line comment

```
/* This is a start and
a matching end */
```

Make comments in .sql files

- Begin your **.sql** files with comments like this:

```
-- fileName.sql
-- …
-- <Student Name>, Section 0, YYYY.MM.DD: <…>
```

- Example:

```
-- ex0_JS.sql
-- Exercise 0
-- Revision History:
-- John Smith, Section 1, 2022.09.06: Created
```

Identify answers in .sql files

- Identify each answer with a comment and print statements, like this:

```
-- 1
Print '*** Question 1 ***';
Print '';

SELECT …
```

SELECT Basics: all columns

- Select all columns from a table:

```
SELECT *
FROM tableName
```

- Example:

```
-- 1
SELECT *
FROM Employee;
```

SELECT Basics: single column

- Select one column from a table:

```
SELECT column
FROM tableName
```

- Example:

```
-- 2
SELECT number
FROM Employee;
```

SELECT Basics: multiple columns

- Select more than one column from a table:

```
SELECT column1, column2 [,…]
FROM tableName
```

- Example:

```
-- 3
SELECT number, reportsTo, campusCode
FROM Employee;
```

Column Alias

- Alias a column:

```
SELECT column1 AS alias1 [, ...]
FROM table
```

...

- Example:

```
-- 4
SELECT lastName AS surname
FROM Person;
```

Column Alias: AS clause can be omitted

- The AS clause is optional and can be omitted:

```
SELECT column1 alias1, column2 [, ...]
FROM table
```

...

- Example:

```
-- 5
SELECT lastName surname, city
FROM Person;
```

Expressions

- Create a calculated column with an expression:

```
SELECT column1, expression
FROM tableName
```

- Example:

```
-- 6
SELECT studentNumber, amount * 1.02
FROM Payment;
```

Expression with a meaningful alias

- Use AS to give the calculated column a meaningful alias:

```
SELECT expression AS alias
FROM tableName
```

- Example:

```
-- 7
SELECT amount * 1.02 AS "penalty"
FROM Payment;
```

SELECT DISTINCT

- Remove duplicate rows from the result set:

```
SELECT DISTINCT column
FROM tableName
```

- Example:

```
-- 8
SELECT DISTINCT studentNumber
FROM Payment;
```

- DISTINCT is NOT a function – it considers all of the values in the SELECT list with each row of the result

```
SELECT DISTINCT studentNumber, invoiceNumber
FROM Payment
```

CONESTOGA
Connect Life and Learning

ORDER BY: one column ascending

- Order by one column in ascending order:

```
SELECT …
ORDER BY column ASC
```

- Example:

```
-- 9
SELECT lastName, firstName
FROM Person
ORDER BY lastName ASC;
```

CONESTOGA
Connect Life and Learning

ORDER BY: ASC is the default

- ASC is the default and therefore is optional (and seldom used in practice):

```
SELECT …
ORDER BY column
```

- Example:

```
-- 10
SELECT lastName, firstName
FROM Person
ORDER BY lastName;
```

CONESTOGA
Connect Life and Learning

ORDER BY: one column descending

- Order by one column in descending order:

```
SELECT …
ORDER BY column DESC
```

- Example:

```
-- 11
SELECT lastName, firstName
FROM Person
ORDER BY lastName DESC;
```

ORDER BY: column2 within column1

- Order by column2 within column1:

```
SELECT …
ORDER BY column1, column2
```

- Example:

```
-- 12
SELECT lastName, firstName
FROM Person
ORDER BY lastName, firstName;
```

ORDER BY: column2 within column1 descending

- Order by column2 within column1 descending:

```
SELECT …
ORDER BY column1 DESC, column2
```

- Example:

```
-- 13
SELECT lastName, firstName
FROM Person
ORDER BY lastName DESC, firstName;
```

CONESTOGA
Connect Life and Learning

ORDER BY: column2 within column1 desc using an alias

- Order by column2 within column1 descending using an alias:

```
SELECT …
ORDER BY alias DESC, column2
```

- Example:

```
-- 14
SELECT lastName AS surname, firstName
FROM Person
ORDER BY surname DESC, firstName;
```

CONESTOGA
Connect Life and Learning

WHERE Clause Syntax

- Syntax:

```
SELECT …
WHERE search-condition
```

- *search-condition* is a mix of:
  - Comparison predicate
  - LIKE predicate
  - Quantified subquery predicate
  - IS NULL predicate
  - and so on, combined using AND, OR and NOT
  - A comparison predicate has the form *column comparisonOperator value*

Comparison Operators

<	**less than**
>	**greater than**
<=	**less than or equal to**
>=	**greater than or equal to**
=	**equal to**
<>	**not equal to** (prior to SQL-92 standard: !=)

CONESTOGA
Connect Life and Learning

WHERE Clause using equal

- SELECT with WHERE using equal (=)

- Example:

```
-- 15
SELECT studentNumber, invoiceNumber
FROM Payment
WHERE amount = 1000.00;
```

Single and Double Quotes

- Use single quotes around literal values in a SQL statement, for example

```
WHERE state = 'NV'
```

- SQL Server lets you omit single quotes for numeric literals
- It permits comparisons of different types by utilizing implicit type conversion
- Can always make this explicit using CAST
- Double quotes are used for nonstandard column names, especially those with blanks – called quoted identifiers

```
SELECT "store number", city, state
```

- Can also use square brackets in SQL Server to denote quoted identifiers.

WHERE Clause using not equal

- SELECT with WHERE using not equal (<>)
- By default, with SQL Server string comparisons are case insensitive
- In SQL Server, CHAR fields are padded with blanks to their full length
- With VARCHAR, SQL Server never stores trailing blanks in a string
- Example:

```
-- 16
SELECT *
FROM Employee
WHERE location <> '4A17';
```

WHERE Clause using greater than

- SELECT with WHERE using greater than (>)
- Inequality comparisons work for ordered types: number, strings, dates and times
- Example:

```
-- 17
SELECT number, campusCode, location
FROM Employee
WHERE number > 6860000
ORDER BY number;
```

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*, *condition2*, or both in the result set”

CONESTOGA
Connect Life and Learning

Compound Expressions: NOT

- WHERE clause conditions can be negated with NOT

```
SELECT ...
WHERE NOT condition
```

- Example:

```
-- 18
SELECT *
FROM School
WHERE NOT code = 'BUS';
```

Compound Expressions: AND

- WHERE clause conditions can be combined with AND

```
SELECT ...
WHERE condition1 AND condition2
```

- Example:

```
-- 19
SELECT studentNumber, amount, transactionDate
FROM Payment
WHERE id > 10 AND paymentMethodID = 3;
```

Compound Expressions: OR

- WHERE clause conditions can be combined with OR

```
SELECT ...
WHERE condition1 OR condition2
```

- Example:

```
-- 20
SELECT *
FROM Person
WHERE firstName = 'John' OR firstName = 'Jon';
```

Precedence

- Precedence refers to the order of evaluation:
  - NOT
  - AND
  - OR
- NOT is highest because it is *unary*
- AND takes precedence over OR by convention
- Operator precedence in computing is like “order of operations” in arithmetic
- The “golden rule”:

  When in doubt, use parentheses

CONESTOGA
Connect Life and Learning

Saving SQL code (.sql) and output (.rpt)

To save SQL code:

- Select **File > Save .sql As…** from the SQL Server Management Studio Express menu

To redirect SQL output to a file:

- Select **Query > Results To > Results to File** (Ctrl+Shift+F) from the SQL Server Management Studio Express menu
- Run the query with **F5** or **!Execute**
- Enter a file name for RPT file (e.g., **ex0.rpt** file to match the SQL code in **ex0.sql**)
- **Result:** RPT file is created and the **Results** window will display messages

CONESTOGA
Connect Life and Learning

Before you submit the .sql and .rpt files

- Before submitting output, check to make sure the quantity of results is reasonable
- No SQL statement in any exercise or assignment will return more than 200 rows; most will return far less
- The output from assignments in this course will fit on between 1 and 5 printed pages
- If your output is larger than this, it is indicative of an error on your interpretation of the question or there is a logic error in your SQL statement

CONESTOGA
Connect Life and Learning

Case Sensitivity

- Case sensitivity is determined at the server level when Microsoft SQL Server is installed
- By default MS SQL Server is case insensitive (like Visual Basic)
- The installer can opt for case sensitivity (like C/C++/C# and Java)
- Our installation is case insensitive

CONESTOGA
Connect Life and Learning

NULL

- The NULL keyword means “undefined”
- NULL is a value distinct from 0 or 0.0, an empty string (''), or a blank string (' ')
- Predicates (conditions) involving NULL evaluate to UNKNOWN
- SQL uses three-valued logic:
  - Anything compared to NULL evaluates to UNKNOWN
  - NOT UNKNOWN yields UNKNOWN
  - TRUE OR UNKNOWN yields TRUE
  - TRUE AND UNKNOWN yields UNKNOWN
  - FALSE OR UNKNOWN yields UNKNOWN
  - FALSE AND UNKNOWN yields FALSE

Which output shows NULL?

- Try this SQL statement:

```
SELECT campusCode, reportsTo, schoolCode
FROM Employee
WHERE number = 2117745
```

- Now try this SQL statement:

```
SELECT campusCode, reportsTo, schoolCode
FROM Employee
WHERE number = 5512736
```

CONESTOGA
Connect Life and Learning

IS NULL

- Use the IS NULL predicate in the query's WHERE clause to select rows with NULL values for particular attributes:

```
SELECT *
FROM Employee
WHERE schoolCode IS NULL
```

IS NOT NULL

- Use the IS NOT NULL in a search condition to retrieve rows with non NULL values:

```
SELECT *
FROM Employee
WHERE schoolCode IS NOT NULL
```

NULL and Equality

- You *CANNOT* use:

  = NULL instead of IS NULL

```
-- this is wrong
SELECT *
FROM Employee
WHERE schoolCode = NULL
```

- You *CANNOT* use:

  != NULL instead of IS NOT NULL

CONESTOGA
Connect Life and Learning

IN

- Instead of OR operator

```
SELECT * FROM Person WHERE firstName = 'John' OR firstName = 'Jon'
```

- Use an IN predicate

```
SELECT * FROM Person WHERE firstName IN ('John' , 'Jon' )
```

- IN can be negated using NOT, as in:

```
NOT IN ('John' , 'Jon')
```

CONESTOGA
Connect Life and Learning

IN for readability

- For readability, IN is preferred when you are working with more than two values:

```
  ... WHERE state IN ('CA', 'CO', 'NV')
```

- Rather than:

```
  ... WHERE state = 'CA' OR state = 'CO' OR state
  = 'NV'
```

- But the two constructions are equivalent

BETWEEN

- Instead of >= AND <= …

```
SELECT number, lastName, firstName
FROM Person
WHERE number >= 1110000 AND number <= 1200000
```

- … you can use a BETWEEN predicate:

```
SELECT number, lastName, firstName
FROM Person
WHERE number BETWEEN 1110000 AND 1200000
```

The comparison is inclusive: A row with number = 1110000 or a row with number = 1200000 would be included in the result set

LIKE

- Use a LIKE predicate to perform basic pattern matching

  Syntax: <expression> LIKE <pattern>

- Literal characters must be present in the given position

```
SELECT firstName, lastName
FROM Person
WHERE firstName LIKE 'John'
ORDER BY firstName, lastName
```

- As with string comparisons, LIKE uses case-insensitive character comparisons with a case-insensitive database

LIKE wildcard character _

- An underscore character (_) matches one arbitrary character in the given position

```
SELECT firstName, lastName
FROM Person
WHERE firstName LIKE 'Joh_'
ORDER BY firstName, lastName
```

CONESTOGA
Connect Life and Learning

LIKE wildcard character %

- A percent character (%) matches zero, one or more characters starting with the given position

```
SELECT firstName, lastName
FROM Person
WHERE firstName LIKE 'Mar%'
ORDER BY firstName, lastName
```

- You can repeat and combine % and _ as needed in a LIKE predicate pattern

CONESTOGA
Connect Life and Learning

NOT (again!)

- You **can** use NOT with the predicates just described:

```
...NOT IN
...NOT BETWEEN
...NOT LIKE
...IS NOT NULL
```

CONESTOGA
Connect Life and Learning