# BI Notes

*James Solomon-Rounce*

*2018-02-24*

# Contents

# Preface

The following notes were taken by me for educational, non-commercial, purposes. If you find the information useful, buy the material/take the course.

# Chapter 1

# Querying Data with Transact-SQL

---

Notes taken during/inspired by the edX course 'Querying Data with Transact-SQL - Microsoft: DAT201x' by Graeme Malcolm and Geoff Allix.

***Course Handouts***

- Course Syllabus
- Getting Started Guide Including Install
- Adventure Works Entity Relationship Diagram
- Adeventure Works db install script
  NOTE: Remember to ensure read only access for everyone to the folder containing the .SQL and other files
- GitHib Repo for course including course materials, slids, labs etc
- A copy of the above materials should they be changed or removed

***Other useful links*** * Transact-SQL Refrence

```
library(DBI)
```

```
## Loading required package: methods
```

```
# creates a connection to the SQL database
# note that "con" will be used later in each connection to the database
con <- DBI::dbConnect(odbc::odbc(),
                      Driver = "SQL Server",
                      Server = "localhost\\SQLEXPRESS",
                      Database = "AdventureWorks",
                      Trusted_Connection = "True")

# Sets knitr to use this connection as the default so we don't need to specify it for every chunk
knitr::opts_chunk$set(connection = "con")
```

## 1.1 Introduction to Transact-SQL

SQL or Structured Query Language was first developed in the 1970s by IBM as a way of interacting with databases. Other vendors have specfic versions of SQL for instance Oracle is PL/SQL, Microsoft's implentation is TSQL or Transact SQL. Both SQL Server (on prem) and Azure SQL Databases (cloud) use the same query language, however Azure is a subset of full TSQL since it some commands relate to local files and

data functions within .NET that relate only to SQL Server. However, as new features are added to Azure, some new commands are being added to Azure.

SQL is a declarative language - you express what it is that you want, the results - rather than specifying the steps taken to acheive that - it is not prodecural like other programming languages, it is set theory based. It is possible to write proceedural elements or steps within TSQL, however if this is ocurring a lot, it is perhaps better done in another language, which may also perform or run better.

In databases, we typically talk about entities - one type of thing - which is contained in each table.
* Entities are represented as relations (tables) * And entity attributes as domains (columns)

Most relationships are normalized, with relationships between primary and foreign keys. This helps to reduce duplication, however there are instances where de-normalised data is desired.

Schemas are namespaces for database objects - is shows a logical layout for all or part of a relational database, *"As part of a data dictionary, a database schema indicates how the entities that make up the database relate to one another, including tables, views, stored procedures, and more."*(see Lucid Chart on Database Schemas). The process of creating a database schema is called data modelling.

When referring to objects in a database, we could use a fully qualified name, such as:

- [server_name.][database_name.][schema_name.]object_name

This is only really relevant for SQL Server, since Azure will only work with one database at a time. Most of the time we typically just use

- schema_name.object_name

The schema name somwtimes be discarded, but it is considered best practice to include this, since there is sometimes some ambiguity about tables e.g. if we have two tables - Product.Order and Customer.Order - which order table is being referred to, that in the customer or product schema?

SQL has a number of SQL Statement Types:

- DML or Data Manipulation Language - SELECT, INSERT, UPDATE, DELETE
- DDL or Data Definition Language - CREATE, ALTER, DROP
- DCL or Data Control Language - GRANT, REVOKE, DENY

The course focuses on DML which is typically for working with data.

**SELECT** statement has a number of possible sub-components:

- FROM [table]
- WHERE [condition for filtering rows]
- GROUP BY [arranges rows by groups]
- HAVING [condition for filtering groups]
- ORDER BY [sorts the output]

Whilst a SQL statement can look like English, it doesn't neccessarily run from top to bottom in terms of the sequence of elements that are run in a query. For instance, the FROM is the first thing that will be run, then the WHERE filter will be run, then we GROUP BY, then SELECT the columns we are interested in and finally ORDER the results. This can be important when running some queries, which will be explored later in the course. When we run a query, it is not an actual table in a database that is return but a set of rows or record set or subset.

### 1.1.1 Data Types

There are a number of different data types in T-SQL as shown below, which are grouped in to a number of different types.

| Exact Numeric | Approximate Numeric | Character |
|---|---|---|
| tinyint | float | char |
| smallint | real | varchar |
| int | | text |
| bigint | | nchar |
| bit | | nvarchar |
| decimal/numeric | | ntext |
| numeric | | |
| money | | |
| smallmoney | | |

Figure 1.1: Transact-SQL Data Types

(#fig:Data Types)

This is more relevant when designing a database, however it is useful to know when querying what data type you have in a broad sense - numeric, data, string and so on - as the types will determine what type of combinations can be combined together in expressions e.g. you can concatenate strings or add numbers together, but you can't concatenate a string and a number together.

Sometimes it is neccessary to convert data from one type to another, there are two ways this could happen

- Implicit conversion - compatible data types are automatically converted
- Explicit conversion - requires an explicit function e.g. CAST / TRY_CAST, STR, PARSE ? TRY_PARSE, CONVERT / TRY_CONVERT

The TRY options will attempt a conversion and if it does not work, a NULL will be returned rather than an error in the non-TRY version.

### 1.1.2 Working with NULLs

There are recognised standards for treating NULL values - ANSI - which says that anythign involving a NULL should return a NULL. There are functions that help us handle NULL values:

- ISNULL(column/variable, value) - Returns *value* (which you can specify) if the column or variable is NULL
- NULLIF(column/variable, value) - Returns NULL if the column or variable is a value - we are almost recoding a non-null to a null
- COALESCE (column/variable1, column/variable2, …) - Returns the value of the first non-NULL column or variable in the list - for instance if contact details, someone might not have an email, so we might want a telephone number, if they don't have that, return an address etc

NULL is used to indicate an unknown or missing value. NULL is **not** equivalent to zero or an empty string.

ISNULL can be used like an IF function in excel, for instance:

> SELECT name, ISNULL(TRY_CAST(size AS Integer), 0) AS NumericSize
> FROM SalesLT.Product;

In this instance, if there is a value that will be returned, if not, the NULL value will be returned as a 0.

We can also use a CASE statement to return a value whilst integrating NULL in to our query, e.g.

```
SELECT name,
    CASE size
      WHEN 'S' THEN 'SMALL'
      WHEN 'M' THEN 'MEDIUM'
      WHEN 'L' THEN 'LARGE'
      WHEN 'XL' THEN 'EXTRA LARGE'
      ELSE ISNULL(Size, 'N/A')
    END AS PRODUCT
  FROM SalesLT.Product;
```

### 1.1.3 Lab Exercises

```
SELECT TOP(10) BusinessEntityID, Name
FROM Sales.Store;
```

Table 1.1: Displaying records 1 - 10

| BusinessEntityID | Name |
|---|---|
| 292 | Next-Door Bike Store |
| 294 | Professional Sales and Service |
| 296 | Riders Company |
| 298 | The Bike Mechanics |
| 300 | Nationwide Supply |
| 302 | Area Bike Accessories |
| 304 | Bicycle Accessories and Kits |
| 306 | Clamps & Brackets Co. |
| 308 | Valley Bicycle Specialists |
| 310 | New Bikes Company |

# Bibliography