SQL DATABASE – SQL SERVER MANAGEMENT

**1. SQL - Microsoft Virtual Academy**

a. Querying with transact-SQL

b. SQL Database fundamental

**2. Power BI - EDX**

a. Analysing and visualising data with Power BI

**Assessment using AdventureWorks DB for Jack Horlock:**

1. What is being sold, who is buying it, and who is selling it (sales people)

2. Implying, when things are happening, is this year better than last year, who sold how much in each month etc.

Some custom and default visuals to have some graphs and answer the questions above using Power BI

Databases loading

* Databases -> Restore database -> Backup files (FULL backup database before using it)
* Path to backup folder = C:\Program Files\Microsoft SQL Server\MSSQL14.SQLEXPRESS\MSSQL\Backup
* External tables -> external non-relational data e.g. CSV, NoSQL, MongoDB, Cassandra etc
* Always backup your database to a .bak file so you can load it up again next time. You will break your database every now and then if you are using it so best to keep a safe copy of it before you do

Database features

* Temporal tables – Keeps the full history of the data changes made and allows easy point in time analysis
* Three-part name = Database + Schema +Table (Don't really need database part if working within it)
  + E.g. [AdventureWorks2016CTP3].[Sales].[SalesOrderDetail]
* dbo - default schema if no schema name provided
* Always look at it from a business POV - understand the business behind the database and find out how everything is connected and the best way to query things
* Memory-optimised table
  + This helps searching through with queries a lot better by improving performance and time taken to run queries
* SQL uses data normalisation but non-relational data don’t have schemas = hence you get what you put in

Keys

* Keys within multiple tables might have different column key names so beware
* PK - Primary Key = the main key for that table (Only one Primary Key per table)
* FK - Foreign Key = Connecting other foreign tables from different schemas by validating that key to another table's PK
* This constrains people using the database work in a specific way e.g. entering customer details before you put an order to the customer (DATA QUALITY ASSURANCE)
* Validates the FK every time = affects the server's performance (Having less FK is good)
* SAP - ERP System - Old tables might not give you all the relevant key data/connections between tables etc.

Data types and Indexing

* char = uses up all the limit
* varchar = only uses the number of characters it needs
* nvarchar = useful for multiple languages / flexible
* One clustered index in each table
  + e.g. SalesOrderID, Date etc (This is used so it is easier to use search for values with a clustered index)
* You can have as many non-clustered indices you want in a table
  + Still a better way to use queries on non-clustered indexes as it doesn't read the entire row by row
* Columnstore index
  + Stored in memory
  + Stores data in columns rather than rows
  + Uses compression to optimise memory usage and performance
  + Can be clustered or non-clustered
    - Clustered columnstore indexes include all columns
    - Only one clustered columnstore index per table

Views and Stored procedures

* Script view/views to look at query to make more sense of the database
* Views = New table that has been made by multiple tables' data columns to be used for further evaluation and exporting the data sets
* Stored procedures will be used a lot for Data Extraction (Any piece of query that you would use more than once)

Non-relational Data

* XML (Extensible Markup Language) – Markup language based data format
  + Processing instruction – What version of xml, character encoding etc.
  + Elements –Objects/keys etc.
  + Text – Actual element values
  + Attributes – order is not enforced
  + Can be used for Azure SQL Database with native XML
* JSON (JavaScript Object Notation)
  + Lightweight data-interchange format
  + Easy for humans to read and write
  + Language independent – flexible to be used in any object-oriented language
  + Can be used to store app data
  + Store into SQL Server using NVARCHAR or DocumentDB

Queries

* Join columns from multiple tables = FROM table1 as a, JOIN table 2 as b, ON a.key = b.key, WHERE condition

Transact-SQL Querying

Transact-SQL (T-SQL)

* Microsoft’s implementation of SQL is T-SQL
* Query language for SQL server and Microsoft Azure
* SQL is declarative, not procedural
  + Describe what you want, not the steps of it
* Relational Databases – multiple tables that are somehow connected to each other
  + Relations – tables
  + Domains – columns
  + Are normalised
  + Relationships are defined using Primary and Foreign Keys (PK and FK)
* [server\_name].[database\_name].[schema\_name].object\_name
  + Best practice to include the schema\_name as a minimum
  + Without schemas, you might have two tables that have the same features for two different things e.g. Sales.Order vs Production.Order
* Data Manipulation Language (DML)
  + Statements for querying and modifying data
  + SELECT, INSERT, UPDATE, DELETE
* SELECT statement
  + SELECT
  + FROM
  + WHERE
  + GROUP BY
  + HAVING
  + ORDER BY
* Data Conversion
  + Implicit conversion – automatically done
  + Explicit conversion
    - CAST(Original AS NEWDATATYPE) - TRY\_CAST
    - CONVERT(NEWDATATYPE, ORIGINAL) - TRY\_CONVERT
    - PARSE – TRY\_PARSE
    - STR
* NULL
  + Anything + NULL is NULL
    - NULL + 2 = NULL
  + NULL = NULL returns false
  + NULL IS NULL returns true
  + ISNULL – returns *value* if column or variable is NULL
  + NULLIF – returns NULL if column or variable is *value*
  + COALESCE – returns the *value* of the first non-NULL column or variable in the list
* Removing duplicates – DISTINCT e.g. SELECT DISTINCT Color FROM Production.Product
* Sorting – ORDER BY e.g. ORDER BY Category, Price DESC;
  + This would sort the categories first then sort the price within each category
* Limiting Factor – SELECT TOP 100 or SELECT TOP 50% or SELECT TOP 100 WITH TIES
  + OFFSET e.g. OFFSET offset\_value ROW(S)
  + FETCH FIRST|NEXT fetch\_value ROW(S) ONLY
* Predicates – used for WHERE or HAVING? Clause
  + = <>
  + IN
  + BETWEEN
  + LIKE
  + AND
  + OR
  + NOT
* Join Concepts
  + Combine rows from multiple tables by specifying matching criteria
    - Usually based on primary key – foreign key relationships
    - Example: Employees AND Sales order (Employee who sold a product)
  + JOIN operator in FROM clause
    - SELECT …

FROM Table1 AS a JOIN Table2 AS b

ON <on\_predicate>; /WHERE <where\_predicate>; 🡨 NOT GOOD TO DO IN WHERE

* + - E.g. ON p.ProductCategoryID = c.ProductCategoryID AND p.Price < c.Price
    - Just add more joins for more tables using JOIN then ON
* Inner Joins
  + Return only rows where a match is found in both input tables
  + Match rows based on attributes supplied in predicate
  + If join predicate operator is =, also known as equi-join
  + Automatically chooses inner join if just join is specified
* Outer Joins
  + Returns all rows from table 1 and any matching rows from table 2
  + One table’s rows are preserved
    - Designated with LEFT, RIGHT, FULL keyword
    - LEFT = Preserve table 1 rows | RIGHT = Preserve table 2 rows | FULL = Preserve both table rows
    - All rows from preserved table output to result set
  + E.g. Returns all employees and if they sold any products, it will match those rows. Else, it will put null for the column from the second table
  + Below shows a LEFT [OUTER] JOIN

* Cross Joins
  + Combine each row from first table with each row from second table
  + All possible combinations output (Cross multiplying concept)
  + Due to cartesian product output, not typically a desired form of join
* Self Joins
  + Compare rows in same table to each other
  + Create two instances of same table in FROM clause
  + E.g. Return all employees and the name of the employee’s manager
    - SELECT …

FROM Table1 AS a

LEFT JOIN Table1 AS b

ON a.ManagerID = b.EmployeeID;

ORDER BY a.ManagerID

* UNION Queries
  + Set of distinct rows combined from all statements
  + Removes duplicates during querying processing (affects performance)
  + UNION ALL retains duplicates during querying processing
  + Column alias are all set up in the first query and the second one is ignored
  + Number of columns must be equal between the two queries
  + Must use same data types – must be compatible for implicit conversion
* Intersect Queries
  + Only distinct rows that appear in both results
  + Kind of like inner join but bringing rows rather than columns
* Except Queries
  + Only distinct rows that appear in the first set but not the second
    - Order in which sets are specified matters (First query is top priority)
* Scalar Function
  + Operate on elements from a single row as inputs, return a single value as output
  + Returns a single scalar value
  + Can be used like an expression in queries
  + May be deterministic or non-deterministic
  + Categories: Conversions, Date and Time, Mathematical, String, System, Text and Image
* Logical Functions
  + Output is determined by comparative logic
    - ISNUMERIC
    - IIF
    - CHOOSE – Chooses the corresponding value for numbers from 1 – n
* Window Functions
  + Functions applied to a window, or set of rows
  + Include ranking, offset aggregate and distribution functions
* Aggregate Functions
  + Functions that operate on set, or rows of data
  + Summarise input rows
  + Without GROUP BY clause, all rows are arranged as one group
* GROUP BY
  + Creates groups for output rows, according to a unique combination of values specified in the GROUP BY clause
  + Calculates a summary value for aggregate functions in subsequent phases
  + Details rows are “lost” after GROUP BY clause is processed
* HAVING
  + Provides a search condition that each group must satisfy
  + WHERE clause is processed before GROUP BY, HAVING clause is processed after GROUP BY
* Subqueries
  + Nested queries = queries within queries
  + Results from the inner query passed to outer query
    - Inner query = expression to the outer query
* Scalar or Multi-Valued
  + Scalar subquery returns single value to outer query
    - Used with SELECT, WHERE, etc.
  + Multi-valued subquery returns multiple values as a single column set to the outer query
    - Used with IN predicate
* Self-Contained or Correlated
  + Most subqueries are self-contained and have no connection with the outer query other than passing it results
  + Correlated subqueries refer to elements of tables used in outer query
    - Dependent on outer query
    - Inner query executes once per outer row
    - May return scalar or multi-valued
* APPLY with Table-Valued Functions
  + CROSS APPLY – applies the right table expression to each row in left table
    - Conceptually like CROSS JOIN between two tables but can correlate data between sources
  + OUTER APPLY – adds rows for those with NULL in columns for right table
    - Conceptually like LEFT OUTER JOIN between two tables
* View
  + Views are name queries with definitions stored in the database
    - Provides abstraction, encapsulation, and simplification
    - Admin perspective – can proved a security layer to the database
  + Views may be referenced in a SELECT statement just like a table
* Temporary Tables
  + Used to hold temporary result sets within a user’s session
  + Created in tempdb and deleted automatically
  + Created with a # prefix
  + Global temporary tables are created with ## prefix
* Table Variables
  + Introduced because temporary tables can cause recompilations
  + Used similarly to temporary tables but scoped to the batch
  + Use only on very small datasets
* Table-Valued Functions
  + TVFs are named objects with definitions stored in the database
  + TVFs return a virtual table to the calling query
  + Unlike views, TVFS support input parameters
* Derived Tables
  + Must have
    - Alias
    - Unique names for all columns
    - Not use ORDER BY (without TOP or OFFSET/FETCH)
    - Not be referred to multiple times in the same query
  + May have
    - Use internal or external aliases for columns
    - Refer to parameters and/or variables
    - Be nested within other derived tables
  + Columns aliases may be defined inline or externally
* Common Table Expressions (CTEs)
  + CTEs are name table expressions defined in a query
  + Recursion
  + Specify a query for the anchor (root) level
  + Use UNION ALL to add a recursive query for other levels
  + Query the CTE with optional MAXRECURSION option
* Grouping sets
  + Grouping sets subclause builds on GROUP BY clauses
  + Allows multiple groupings to be defined in same query
* ROLLUP and CUBE
  + ROLLUP provides shortcut for defining grouping sets with combinations that assume input columns form a hierarchy
  + CUBE provides shortcut for defining grouping sets in which all possible combinations of grouping sets created
* Identifying Groupings in Results
  + Multiple grouping sets present a problem in identifying the source of each row in the result set
  + NULLs could come from the source data or could be a placeholder in the grouping set
  + The GROUPING\_ID function provides a method to make a row with a 1 or 0 to identify which grouping set for the row
* Pivoting Data
  + Pivoting data is rotating data from a rows-based orientation to a columns-based orientation
  + Distinct values from a single column are projected across as headings for other columns
    - May include aggregation
* Unpivoting Data
  + Unpivoting data is rotating data from a columns-based orientation to a rows-based orientation
  + Spreads or splits values from one source row into one or more target rows
  + Each source row becomes one or more rows in results set based on number of columns being pivoted
* Batches
  + Batches are sets of commands sent to SQL Server as a unit
  + Batches determine variable scope, name resolution
  + To separate statements into batches, use a separator
    - SQL Server tools use the GO keyword
    - GO is not a T-SQL command
    - GO[count] executes the batch the specified number of times
* Comments
  + Marks T-SQL code as a comment - /\* \*/ OR --
  + To help the collaborator to understand the reasons behind the queries you used
* Variables
  + Variables are objects that allow storage of a value for use later in the same batch
  + Variables are defined with the DECLARE keyword
  + Variables are always local to the batch in which they’re declared and go out of scope when the batch ends
* Conditional Branching
  + IF…ELSE uses a predicate to determine the flow of the code
* Looping
  + WHILE enables code to execute in a loop
  + Statements in the WHILE block repeat as the predicate evaluates to TRUE
  + The loop ends when the predicate evaluates to FALSE or UNKNOWN
  + Execution can be altered by BREAK or CONTINUE
* Stored Procedures
  + Database objects that encapsulate Transact-SQL code
  + Can be parameterized
  + Executed with the EXECUTE command
* Error and Error Messages
  + Elements of Database Engine Errors
    - Error number
    - Error message
    - Severity
    - State
    - Procedure
    - Line Number
  + In SQL Server (NOT AZURE SQL DB)
  + Error messages are in sys.messages
  + You can add custom messages using sp\_addmessage
* Raising Errors
  + The RAISEERROR Command
    - Raise a user-define error in sys.messages (SQL Server only)
    - Raise an explicit error message, severity, and state (SQL Server and Azure SQL DB only)
  + The THROW Command
    - Replacement for RAISEERROR
    - Throw explicit error number, message, and state (severity is 16)
    - Re-throw existing error
* Catching and Handling Errors
  + Use a TRY…CATCH block
  + Handle errors in the CATCH block
    - Get error information
    - Execute custom correction or logging code
    - Re-throw the original error, or throw a custom error
* Transactions
  + A transaction is a group of tasks defining a unit of work
  + The entire unit must succeed or fail together – no partial completion is permitted
  + Individual data modification statements are automatically treated as standalone transactions
  + SQL Server uses locking mechanisms and the transaction log to support transactions
* Implementing Explicit Transactions
  + Use BEGIN TRANSACTION to start a transaction
  + USE COMMIT TRANSACTION to complete a transaction
  + UES ROLLBACK TRANSACTION to cancel a transaction
    - Or enable XACT\_ABORT to automatically rollback on error
  + USE @@TRANCOUNT and XACT\_STATE() to check transaction status