Key Points and Additional Reading

Module 1: Introduction to Transact-SQL

KEY POINTS	ADDITIONAL READING
Getting Started with Transact-SQL	
• Transact-SQL is the language used to query data in Microsoft SQL Server and Azure SQL Database. • Data is stored in tables, which may be related to one another through common key fields. • Objects in a database are organized into schemas. • The fully qualified naming syntax for an object is *server_name*.*database_name*.*schema_name*.*object_name*, but in most cases you can abbreviate this to *schema_name.object_name.*	Throughout this course, links to specific sections in the Transact-SQL Reference documentation will be provided to supplement the course materials. Take a look at the overview page for this reference.
The SELECT Statement	
• Use the SELECT statement to retrieve a rowset of data from tables and views in a database. • SELECT statements are written with the following clauses: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY. • However, the query engine processes the clauses in this order: FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY. • In the SELECT clause, you can use * to return all columns, but generally you should specify explicit columns. • You can specify expressions in the SELECT clause to return the results of calculations. • You can use the AS keyword to specify aliases for columns in the rowset returned by the SELECT statement.	Review the documentation for SELECT in the Transact-SQL Reference.
Working with Data Types	
• Transact-SQL supports a wide range of data types, which can be broadly categorized as **exact numeric**, **approximate numeric**, **character**, **date/time**, **binary**, and **other** (which includes specialized data types for handling data such as XML and spatial data). • Some data types are compatible, and values can be implicitly converted between them. Conversion between other data types requires the use of explicit conversion functions.	Review the documentation for Data Types and Conversion Functions in the Transact-SQL Reference.
Working with NULLs	
• NULL is used to indicate an unknown or missing value. NULL is **not** equivalent to zero or an empty string. • Arithmetic or string concatenation operations involving one or more NULL operands return NULL. For example, 12 + NULL = NULL. • If you need to compare a value to NULL, use the **IS** operator instead of the **=** operator. • The **ISNULL** function returns a specified alternative value for NULL columns and variables. • The **NULLIF** function returns NULL when a column or variable contains a specified value. • The **COALESCE** function returns the first non-NULL value in a specified list of columns or variables).	Review the documentation for the ISNULL function and Expressions in the Transact-SQL Reference.

Module 2: Querying Tables with SELECT

KEY POINTS	ADDITIONAL READING
Eliminating Duplicates and Sorting Results	
• By default, the SELECT statement returns all rows. If multiple rows contain the same values for every column, they are duplicated in the results. Using the DISTINCT keyword eliminates duplicates, ensuring that only one row for each distinct combination of column values is returned. • The order of rows in the result of a SELECT statement is not guaranteed unless you explicitly specify one or more columns in an ORDER BY clause. You can specify sort direction as ASC (the default) or DESC. • You can combine the ORDER BY clause with the TOP keyword to restrict the results so that they include only the top n rows (where n is the number or percentage of rows you want to return). • You can implement a query to retrieve a specified "page" of results by using the OFFSET and FETCH keywords with the ORDER BY clause.	Review the documentation for the SELECT and ORDER BY clauses in the Transact-SQL Reference.
Filtering and Using Predicates	
• Use the WHERE clause to filter the results returned by a SELECT query based on a search condition. • A search condition is composed of one or more predicates. • Predicates include conditional operators (such as =, >, and <), IN, LIKE, and NOT. • You can use AND and OR to combine predicates based on Boolean logic.	Review the documentation for WHERE and Search Condition in the Transact-SQL Reference.

Wildcard character	Description	Example
%	Any string of zero or more characters.	WHERE title LIKE '%computer%' finds all book titles with the word 'computer' anywhere in the book title.
_ (underscore)	Any single character.	WHERE au_fname LIKE '_ean' finds all four-letter first names that end with ean (Dean, Sean, and so on).
[ ]	Any single character within the specified range ([a-f]) or set ([abcdef]).	WHERE au_lname LIKE '[C-P]arsen' finds author last names ending with arsen and starting with any single character between C and P, for example Carsen, Larsen, Karsen, and so on. In range searches, the characters included in the range may vary depending on the sorting rules of the collation.
[^]	Any single character not within the specified range ([^a-f]) or set ([^abcdef]).	WHERE au_lname LIKE 'de[^l]%' all author last names starting with de and where the following letter is not l.

Module 3: Querying Multiple Tables with Joins

KEY POINTS	ADDITIONAL READING
Introduction to Joins	
• Joins are used to match rows in one table to rows in another table. • The query engine supports two ways to define joins: the ANSI SQL-92 syntax (in which the join is specified in the FROM clause) and the older ANSI SQL-89 syntax (in which the join is specified in the WHERE clause). The ANSI SQL-92 syntax is the preferred approach.	Review the documentation on Join Fundamentals in SQL Server 2008 R2 Books Online.
Inner Joins	
• Inner joins return only rows where a match can be found in both tables. • Inner joins that match rows based on columns containing the same value in both tables are sometimes referred to as equi-joins.	Review the documentation on Using Inner Joins in SQL Server 2008 R2 Books Online.
Outer Joins	
• Use a Left Outer Join to include all rows from the first table and values from matched rows in the second table. Columns in the second table for which no matching rows exist are populated with NULLs. • Use a Right Outer Join to include all rows from the second table and values from matched rows in the first table. Columns in the first table for which no matching rows exist are populated with NULLs. • Use a Full Outer Join to include all rows from the first and second tables. Columns in the either table for which no matching rows exist are populated with NULLs.	Review the documentation on Using Outer Joins in SQL Server 2008 R2 Books Online.
Cross Joins	
• A cross join returns a Cartesian product that includes every combination of the selected columns from both tables. • While not commonly used in typical application processing, cross joins can be useful in some specialized scenarios - such as generating test data.	Review the documentation on Using Cross Joins in SQL Server 2008 R2 Books Online.
Self Joins	
• A self-join is an inner, outer, or cross join that matches rows in a table to other rows in the same table. • When defining a self-join, you must specify an alias for at least one instance of the table being joined.	Review the documentation on Using Self-Joins in SQL Server 2008 R2 Books Online.

Module 4: Using Set Operators

KEY POINTS	ADDITIONAL READING
UNION Queries	
• Use UNION to combine the row sets returned by multiple queries. • Each unioned query must return the same number of columns with compatible data types. • By default, UNION eliminates duplicate rows. Specify the ALL option to include duplicates (or to avoid the overhead of checking for duplicates when you know in advance that there are none).	Review the documentation on UNION in the Transact-SQL Reference.
INTERSECT and EXCEPT Queries	
• Use INTERSECT to return only rows that are returned by both queries. • Use EXCEPT to return rows from the first query that are not returned by the second query.	Review the documentation on EXCEPT and INTERSECT in the Transact-SQL Reference.

Module 5: Using Functions and Aggregating Data

KEY POINTS	ADDITIONAL READING
Introduction to Functions	
• Scalar functions return a single value based on zero or more input parameters. • Logical functions return Boolean values (true or false) based on an expression or column value. • Window functions are used to rank rows across partitions or "windows". Window functions include RANK, DENSE_RANK, NTILE, and ROW_NUMBER. • Aggregate functions are used to provide summary values for mulitple rows - for example, the total cost of products or the maximum number of items in an order. Commonly used aggregate functions include SUM, COUNT, MIN, MAX, and AVG.	Review the documentation on Built-In Functions in the Transact-SQL Reference.
Grouping Aggregated Data	
• You can use GROUP BY with aggregate functions to return aggregations grouped by one or more columns or expressions. • All columns in the SELECT clause that are not aggregate function expressions must be included in a GROUP BY clause. • The order in which columns or expressions are listed in the GROUP BY clause determines the grouping hierarchy. • You can filter the groups that are included in the query results by specifying a HAVING clause.	Review the documentation on GROUP BY and HAVING in the Transact-SQL Reference *Note that this module has discussed only simple GROUP BY queries. General GROUP BY clauses that include grouping sets, ROLLUP, or CUBE operators are covered later in the course.*

Module 6: Using Subqueries and APPLY

KEY POINTS	ADDITIONAL READING
Using Subqueries	
• Subqueries are Transact-SQL queries nested within an outer query. • Scalar subqueries return a single value. • Multi-valued subqueries return a single-column rowset.	Review the documentation on Subquery Fundamentals in SQL Server 2008 R2 Books Online. *Note that while this documentation is based on a previous release of SQL Server, it is still relevant.*
Using Correlated Subqueries	
• Correlated subqueries reference objects in the outer query.	Review the documentation on Correlated Subqueries in SQL Server 2008 R2 Books Online. *Note that while this this documentation is based on a previous release of SQL Server, it is still relevant.*
Using Apply	
• The APPLY operator enables you to execute a table-valued function for each row in a rowset returned by a SELECT statement. Conceptually, this approach is similar to a correlated subquery. • CROSS APPLY returns matching rows, similar to an inner join. OUTER APPLY returns all rows in the original SELECT query results with NULL values for rows where no match was found.	Review the documentation on Using Apply in SQL Server 2008 R2 Books Online. *Note that while this documentation is based on a previous release of SQL Server, it is still relevant.*

Module 7: Using Table Expressions

KEY POINTS	ADDITIONAL READING
Querying Views	
• Views are database objects that encapsulate SELECT queries. • You can query a view in the same way as a table, however there are some considerations for updating them.	Review the documentation on CREATE VIEW in the Transact-SQL Language Reference.
Using Temporary Tables and Table Variables	
• Temporary tables are prefixed with a # symbol and stored in a temporary workspace (the tempdb database in SQL Server). • Temporary tables are automatically deleted when the session in which they were created ends. • Excessive use of temporary tables can negatively affect overall database server performance. • Table variables are prefixed with a @ symbol and are stored in memory. • Table variables are scoped to the batch in which they are created. • Table variables work best with small sets of data.	Review the documentation for the table data type in the Transact-SQL Language Reference.
Querying Table-Valued Functions	
• Table-Valued Functions (TVFs) are functions that return a rowset. • TVFs can be parameterized.	Review the documentation on Table-Valued User-Defined Functions in SQL Server 2008 R2 Books Online. *Note that while this documentation is based on a previous release of SQL Server, it is still relevant.*
Using Common Table Expressions	
• A derived table is a subquery that generates a multicolumn rowset. You must use the AS clause to define an alias for a derived query. • Common Table Expressions (CTEs) provide a more intuitive syntax or defining rowsets than derived tables, and can be used mulitple times in the same query. • You can use CTEs to define recursive queries.	Review the documentation on WITH common_table_expression in the Transact-SQL Language Reference.

Module 8: Grouping Sets and Pivoting Data

KEY POINTS	ADDITIONAL READING
Grouping Sets	
• Use GROUPING SETS to define custom groupings. • Use ROLLUP to include subtotals and a grand total for hierarchical groupings. • Use CUBE to include all possible groupings.	Review the documentation on GROUP BY in the Transact-SQL Language Reference.
Pivoting Data	
• Use PIVOT to re-orient a rowset by generating mulitple columns from values in a single column. • Use UNPIVOT to re-orient mulitple columns in a an existing rowset into a single column.	Review the documentation on Using PIVOT and UNPIVOT in SQL Server 2008 R2 Books Online *Note that while this documentation is based on a previous release of SQL Server, it is still relevant.*

Module 9: Modifying Data

KEY POINTS	ADDITIONAL READING
Inserting Data	
• Use the INSERT statement to insert one or more rows into a table. • When inserting explicit values, you can omit identity columns, columns that allow NULLs, and columns on which a default constraint is defined. • Identity columns generate a unique integer identifier for each row. You can also use a sequence to generate unique values that can be used in multiple tables.	Review the documentation on INSERT, IDENTITY in the Transact-SQL Language Reference, and Sequence Numbers in SQL Server Books Online.
Updating and Deleting Data	
• Use the UPDATE statement to modify the values of one or more columns in specified rows of a table. • Use the DELETE statement to delete specified rows in a table. • Use the MERGE statement to insert, update, and delete rows in a target table based on data in a source table.	Review the documentation on UPDATE, DELETE, and MERGE in the Transact-SQL Language Reference.

Module 10: Programming with Transact-SQL

KEY POINTS	ADDITIONAL READING
Batches, Comments and Variables	
• A batch defines a group of Transact-SQL command submitted by a client application for execution. Some commands can only be executed at the start of a new batch, and variable values cannot span batches. • Use comments to document your Transact-SQL code. Inline comments are prefixed by --, and multi-line comment blocks are enclosed in /* and */. • Declare variables by using the DECLARE keyword, specifying a name (prefixed with @) and a data type. You can optionally assign an initial value. • Assign values to variables by using the SET keyword or in a SELECT statement.	Review the documentation on --, /*..*/, and Variables in the Transact-SQL Language Reference.
Conditional Branching	
• Use the IF keyword to execute a task based on the results of a conditional test. • Use an ELSE clause if you need to execute an alternative task if the conditional test returns false. • Enclose multiple statements in an IF or ELSE clause between BEGIN and END keywords.	Review the documentation on IF...ELSE and BEGIN...END in the Transact-SQL Language Reference.
Looping	
• Use a WHILE loop if you need to repeat one or more statements until a specified condition is true. • Use BREAK and CONTINUE to exit or restart the loop. • Avoid using loops to iteratively update or retrieve single records - in most cases, you should use set-based operations to retrieve and modify data.	Review the documentation on WHILE, BREAK, and CONTINUE in the Transact-SQL Language Reference.
Stored Procedures	
• Use stored procedures to encapsulate Transact-SQL code in a reusable database objects. • You can define parameters for a stored procedure, and use them as variables in the Transact-SQL code it contains. • Stored procedures can return rowsets (usually the results of a SELECT statement). They can also return output parameters, and they always return a return value, which is used to indicate status.	Review the documentation on Stored Procedures in SQL Server Books Online.

Module 11: Error Handling and Transactions

KEY POINTS	ADDITIONAL READING
Raising Errors	
• System errors have pre-defined numbers, messages, severity levels, and other characteristics that you can use to troubleshoot issues. • You can use RAISERROR and THROW to raise custom errors.	Review the documentation on RAISERROR and THROW in the Transact-SQL Language Reference.
Catching and Handling Errors	
• Use TRY...CATCH blocks in your Transact-SQL code to catch and handle exceptions. • A common exception handling pattern is to log the error, and then if the operation cannot be completed successfully, throw it (or a new custom error) to the calling application.	Review the documentation on TRY...CATCH in the Transact-SQL Language Reference.
Implementing Transactions	
• Transactions are used to protect data integrity by ensuring that all data changes within a transaction succeed or fail as a unit. • Individual Transact-SQL statements are inherently treated as transactions, and you can define explicit transactions that encompass mulitple statements. • Use the BEGIN TRANSACTION, COMMIT TRANSACTION, and ROLLBACK TRANSACTION statements to manage transactions. • Enable the XACT_ABORT option to automatically rollback all transactions if an exception occurs. • Use the @@TRANCOUNT system variable and XACT_STATE system function to determine transaction status.	Review the documentation on Transaction Statements, @@TRANCOUNT, and XACT_STATE in the Transact-SQL Language Reference. For a detailed explanation of how the transaction log is used for data updates and recovery, see the Technet Magazine article Understanding Logging and Recovery in SQL Server by Paul S. Randall.

Course Wrap-Up

Congratulations! You've completed this course on Querying Transact-SQL.

CONTINUE LEARNING

Transact-SQL is a fundamental skill for working with SQL Server and Azure SQL Database. To learn more about these database platforms, you can continue your studies by taking online courses at Microsoft Virtual Academy.

CONSIDER CERTIFICATION

The Microsoft Certified Professional program validates skills with Microsoft technologies and awards industry-recognized certifications. This course can help you prepare for exam 70-461: Querying Microsoft SQL Server, which is a required exam for the Microsoft Certified Solutions Associate (MCSA): SQL Server certification.

Note: While this course covers many of the core objectives measured by Exam 70-461, the exam may test some additional objectives beyond those covered in this course. Before taking the exam, review the skills measured and ensure that you have supplemented your learning on this course with additional information from SQL Server Books Online and other materials, such as T-SQL Querying (Microsoft Press, 2015), by Itzik Ben-Gan, Adam Machanic, Dejan Sarka, and Kevin Farlee. This book gives database developers and administrators a detailed look at the internal architecture of T-SQL and is the comprehensive programming reference for T-SQL querying.