[edX Home Page](https://courses.edx.org/dashboard)

Microsoft: DAT201x

**Querying Data with Transact-SQL**

Module 1

* 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.*
* 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**.
* 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.
* 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.
* 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).

Module 2

* 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.
* 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.

Module 3

* **JOIN**s are used to match rows in one table to rows in another table.
* The query engine supports two ways to define **JOIN**s:
  + ANSI SQL-92 syntax (in which the join is specified in the **FROM** clause) is the preferred approach.
  + ANSI SQL-89 syntax (in which the join is specified in the **WHERE** clause)
* **INNER JOIN**s return only rows where a match can be found in both tables.
* **INNER JOIN**s that match rows based on columns containing the same value in both tables are sometimes referred to as equi-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.
* 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 JOIN**s can be useful in some specialized scenarios - such as generating test data.
* 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.

Module 4

* Use **UNION** to combine the rowsets returned by multiple queries.
* Column **aliases** – must be expressed in first query
* Each **UNION** query must return **the same number of columns** with **compatible data types**.
* By default, **UNION** eliminates duplicate rows. Specify the **UNION** **ALL** option to include duplicates (or to avoid the overhead of checking for duplicates when you know in advance that there are none).
* 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.

Examples:

SELECT FirstName, LastName, ‘Employee’ AS Type

FROM SalesLT.Employees

**UNION**

SELECT FirstName, LastName, ‘Customer’

FROM SalesLT.Customers

ORDER BY LastName;

SELECT FirstName, LastName

FROM SalesLT.Customers

**INTERSECT**

SELECT FirstName, LastName

FROM SalesLT.Employees

SELECT FirstName, LastName

FROM SalesLT.Customers

**EXCEPT**

SELECT FirstName, LastName

FROM SalesLT.Employees