### Data types

- **timestamp**: Used to store a result set for processing at a later time. Is a special data type that can contain a reference to a cursor. Max 2,147,483,647 bytes.
- **binary numbers within a database.**
- **nchar**: Variable-length Unicode data. Max 2,147,483,647 characters
- **nvarchar**: Variable-length non-Unicode data. Max 8000 characters
- **varchar**: Fixed-length non-Unicode character. Max 2,147,483,647 characters
- **smalldatetime**: Returns the exact time that an action occurred. Min 4, max 19 bytes
- **smallint**: Integer data type. Min 2, max 4 bytes
- **bigint**: A 10-byte integer data type. Min 8, max 8 bytes
- **float**: A floating-point number data type. Min 4, max 3.40E + 38
- **decimal**: Represents a numeric data type. Min: 0, max: 38
- **money**: A floating decimal number data type. Min: 0, max: 19 digits
- **numeric**: A floating decimal number data type. Min: 0, max: 38
- **bit**: A binary data type. Min: 1, max: 1 byte
- **datetime**: A 8-byte date and time data type. Max 4,800,000,000,000,000,000

### Character Strings

- **binary**: Character strings. Max 8000 bytes
- **nchar**: Variable-length Unicode data. Max 8000 characters
- **nvarchar**: Variable-length non-Unicode data. Max 8000 characters
- **varchar**: Fixed-length non-Unicode character. Max 8000 characters

### System Functions (T-SQL)

- **ABS**: Returns the absolute (positive) value of a numeric expression.
- **ACOS**: Returns the angle, in radians, whose cosine is the specified float expression; also called arccosine.
- **ASIN**: Returns the angle, in radians, whose sine is the specified float expression.
- **ATAN2**: Returns the angle, in radians, whose tangent is the specified float expression. This is also called arctangent.
- **ATAN**: Returns the angle, in radians, whose tangent is the specified float expression; also called arctangent.
- **COS**: Returns the trigonometric cosine of the specified angle, in radians.
- **COT**: Returns the trigonometric cotangent of the specified angle, in radians.
- **DEGREES**: Returns the corresponding angle in degrees for an angle specified in radians.
- **EXP**: Returns the exponential value of the specified float expression.
- **FLOOR**: Returns the largest integer less than or equal to the specified numeric expression.
- **LOG**: Returns the natural logarithm of the specified float expression.
- **LOG10**: Returns the base-10 logarithm of the specified float expression.
- **PI**: Returns the constant value of PI.
- **POWER**: Returns the value of the specified expression raised to the specified power.
- **RADIANS**: Returns radians of the numeric expression, in degrees.
- **RAND**: Returns a random float value from 0 through 1.
- **ROUND**: Rounds the specified numeric expression to the specified number of decimal places.
- **SIGN**: Returns the positive (+1), zero (0), or negative (-1) sign of the specified expression.
- **SIN**: Returns the trigonometric sine of the specified angle, in radians.
- **SQRT**: Returns the square root of the specified float expression.
- **SQUARE**: Returns the square of the specified float value.
- **TAN**: Returns the tangent of the specified expression.
SELECT (T-SQL)
The full syntax of the SELECT statement is complex, but the main clauses can be summarized as:

- **SELECT** [DISTINCT] [(TOP int | TOP int PERCENT) | ALL] columns
- INTO new_table
- FROM table_source
table_name [LEFT | RIGHT | FULL | OUTER JOIN] table_source2

Example:
```
SELECT * FROM Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
```

ALTER TABLE (T-SQL)
Modifies a table definition by adding, dropping or altering columns and constraints, or by disabling or enabling constraints and triggers. The full syntax is complex, but the main clauses can be summarized as:

- **ALTER TABLE** table_name
- **ALTER** [ADD | DROP | RENAME COLUMN | MODIFY COLUMN | ENABLE | DISABLE] column

Example:
```
ALTER TABLE Orders
ADD new_column
```

DELETE (T-SQL)
The full syntax of the DELETE statement is complex, but the main clauses can be summarized as:

- **DELETE** [FROM] table_name
- **WHERE** search_condition

Example:
```
DELETE FROM Orders
WHERE OrderDate < '2020-01-01'
```

CREATE TABLE (T-SQL)
Creates a new table. The full syntax is complex, but the main clauses can be summarized as:

- **CREATE TABLE** ([database_name].[owner.] table_name
- (column_definition) [column_name
- ]...]

Example:
```
CREATE TABLE Employees
    (EmployeeID int PRIMARY KEY,
     FirstName varchar(50),
     LastName varchar(50)
    )
```

ALTER COLUMN (T-SQL)
Modifies a column's definition by changing its data type, name, or constraints. The full syntax is complex, but the main clauses can be summarized as:

- **ALTER TABLE** table_name
- **ALTER COLUMN** column_name [NEW_NAME new_name]
- [NEW_DATA_TYPE new_data_type]
- [CONSTRAINT {<table_constraint>}]

Example:
```
ALTER TABLE Employees
ALTER COLUMN EmployeeID INT
```

DELETE COLUMN (T-SQL)
Removes a column from a table. The full syntax is complex, but the main clauses can be summarized as:

- **ALTER TABLE** table_name
- **DROP COLUMN** column_name

Example:
```
ALTER TABLE Employees
DROP COLUMN LastUpdate
```

CREATE INDEX (T-SQL)
Creates an index on one or more columns. The full syntax is complex, but the main clauses can be summarized as:

- **CREATE INDEX** index_name [ON table_name [column_list]]
- [CONSTRAINT {<table_constraint>}]

Example:
```
CREATE INDEX idx_Employees_EmployeeID
ON Employees (EmployeeID)
```

CHECK CONSTRAINT (T-SQL)
Verifies that a table's constraints are satisfied. The full syntax is complex, but the main clauses can be summarized as:

- **CHECK** table_name
- [CONSTRAINT {<table_constraint>}]

Example:
```
CHECK table_Employees
```

Creating / Altering other objects

Stored procedures:
```
CREATE PROCEDURE name AS <sql_statement>
```

Indexes:
```
CREATE INDEX index_name ON table_name (column)
CREATE UNIQUE INDEX index_name ON table_name (column)
```

Views:
```
CREATE VIEW view_name [(column1,...)] AS <sql_statement>
```

Triggers:
```
CREATE TRIGGER name ON table_name FOR INSERT, UPDATE, DELETE AS <sql_statement>
```

Functions:
```
CREATE FUNCTION name AS <sql_expression>
```

Sargability
Select statements only use indexes for columns used in sargable expressions. Green = fastest expression.

- LIKE ‘abc%’
- BETWEEN 10 and 20
- NOT IN (1, 2, 3)

Checklist for fast queries
- Avoid non-sargable WHERE-clauses. If possible rewrite them to sargable ones.
- In the WHERE-clause use the least true AND expression first.
- Avoid using OR in the WHERE-clause if not all columns have an index.
- Avoid using UNION if UNION ALL also does the trick.
- Avoid using IN when BETWEEN is also possible.
- Avoid using BETWEEN in the WHERE-clause. If possible use LIKE instead.
- Sometimes consider rewriting a query using OR to multiple queries combined with a UNION ALL.
- Don’t use ORDER BY if you don’t really need it.
- Keep the width and number of sorted columns to a minimum.
- Keep the number of rows to a minimum.
- When selecting a specific column often consider using a clustered index.
- In case of using HAVING try to minimize the amount of rows using a WHERE clause.
- In case using LIKE on CHAR or VARCHAR columns quite often consider using the full-text search option.
- In case using GROUP BY without an aggregate function try using DISTINCT instead.
- Avoid using variables in a WHERE-clause in case the query is located in a batch-file.

Checklist for creating indexes
- Create indexes on the highly selective columns that are used in the WHERE-clause.
- Create indexes on all columns that are used in the WHERE clause in case OR is used.
- Create at least a clustered index on every table. Generally use the column that monotonically increases.
- Create indexes columns that are frequently accessed by WHERE, ORDER BY, GROUP BY, TOP, and DISTINCT.
- Only add indexes that will be used frequently.
- Avoid adding too much indexes on dynamic tables (subject to many INSERT, UPDATE, or DELETEs).
- For static tables use a FILI因子or PAD_INDEX of 100. For dynamic tables use a lower FILI因子CTOR.
- To identify additional indexes use the SS Profiler Create Trace Wizard and trace “Identify Scans of Large Tables”
- Avoid adding indexes twice.