

# 1. Introduction to T-SQL

## Basics of T-SQL

- **Question:** What is T-SQL, and how is it different from standard SQL?

**Answer:** T-SQL (Transact-SQL) is Microsoft's extension of SQL used in SQL Server. It adds procedural programming capabilities like variables, loops, error handling, and conditional logic. It includes enhancements for querying and modifying data, such as built-in functions, exception handling, and transaction control.

---

## 2. T-SQL Data Types

### Data Type Selection

- **Question:** How do you choose appropriate data types for columns in T-SQL?

**Answer:** You select data types based on the kind of data being stored:

- **INT** for integer values.
- **DECIMAL** or **FLOAT** for precise or floating-point numbers.
- **VARCHAR** or **NVARCHAR** for variable-length text.
- **DATETIME** for date and time. Choosing appropriate data types optimizes performance and minimizes storage requirements.

### Handling NULL Values

- **Question:** How does T-SQL handle `NULL` values, and why are they significant?

**Answer:** `NULL` represents an unknown or missing value. In T-SQL, `NULL` values are not equal to any other value (even `NULL` itself). Functions like `IS NULL` or `ISNULL()` are used to handle `NULL` values in queries.

---


## 3. Basic Queries and Joins

### Writing Basic Queries

- **Question:** How do you write a basic `SELECT` query in T-SQL to retrieve specific columns from a table?

**Answer:** A basic `SELECT` query retrieves columns from a table:

```
sql
SELECT FirstName, LastName FROM Employees WHERE Department = 'HR';
```

 Copy code

### JOIN Types

- **Question:** What are the different types of joins in T-SQL, and how are they used?

**Answer:** T-SQL supports several types of joins:

- **INNER JOIN:** Retrieves records with matching values in both tables.
- **LEFT JOIN:** Returns all records from the left table and matching records from the right table (or `NULL` if no match).
- **RIGHT JOIN:** Returns all records from the right table and matching records from the left table.
- **FULL JOIN:** Retrieves all records when there is a match in either table.

sql

 Copy code

```
SELECT E.FirstName, D.DepartmentName FROM Employees E LEFT JOIN Departments D ON  
E.DepartmentId = D.DepartmentId;
```

---

## 4. T-SQL Functions

### Aggregate Functions

- **Question:** What are aggregate functions in T-SQL, and when would you use them?

**Answer:** Aggregate functions perform calculations on a set of values and return a single result. Common aggregate functions include:

- **COUNT():** Counts rows.
- **SUM():** Sums a numeric column.
- **AVG():** Averages values.
- **MIN() and MAX():** Returns the minimum or maximum value.

sql

 Copy code

```
SELECT COUNT(*), AVG(Salary) FROM Employees WHERE Department = 'HR';
```

### Scalar Functions

- **Question:** What are scalar functions, and how are they used in T-SQL?

**Answer:** Scalar functions operate on a single value and return a single value. T-SQL has built-in scalar functions for string manipulation, date and time operations, and mathematical calculations. Example functions include `LEN()`, `GETDATE()`, `CONVERT()`, and `ABS()`.

sql

 Copy code

```
SELECT FirstName, LEN(FirstName) AS NameLength FROM Employees;
```

---

## 5. Subqueries and Common Table Expressions (CTEs)

### Subqueries

- **Question:** What is a subquery in T-SQL, and how is it used?

**Answer:** A subquery is a query nested inside another query. It can return single or multiple values and is used in `WHERE`, `FROM`, or `SELECT` clauses. Subqueries can also be correlated or non-correlated.

sql

 Copy code

```
SELECT FirstName, LastName FROM Employees WHERE DepartmentId = (SELECT DepartmentId  
FROM Departments WHERE DepartmentName = 'HR');
```

### Common Table Expressions (CTEs)

- **Question:** What is a CTE, and how does it differ from a subquery?

**Answer:** A CTE is a temporary result set defined within a `WITH` clause, used for simplifying complex queries. CTEs can reference themselves for recursive queries and improve readability over subqueries.

sql

 Copy code

```
WITH EmployeeCTE AS ( SELECT FirstName, LastName, ManagerId FROM Employees WHERE  
ManagerId IS NOT NULL ) SELECT * FROM EmployeeCTE;
```


---

## 6. T-SQL Control of Flow

### IF...ELSE

- **Question:** How do you use the IF...ELSE construct in T-SQL?


**Answer:** IF...ELSE is used for conditional execution of T-SQL statements. Based on the result of the condition, specific code is executed.

```
sql  Copy code  
  
IF EXISTS (SELECT 1 FROM Employees WHERE Salary > 100000) PRINT 'High Salary  
Detected'; ELSE PRINT 'No High Salary Found';
```

### WHILE Loop

- **Question:** How do you create a loop in T-SQL using WHILE ?

**Answer:** WHILE is used to repeatedly execute a block of T-SQL statements as long as a condition remains TRUE .

```
sql  Copy code  
  
DECLARE @Counter INT = 1; WHILE @Counter <= 10 BEGIN PRINT @Counter; SET @Counter =  
@Counter + 1; END;
```


---

## 7. Error Handling in T-SQL

### TRY...CATCH

- **Question:** How do you handle errors in T-SQL using TRY...CATCH ?

**Answer:** TRY...CATCH is used to handle errors during the execution of T-SQL code. If an error occurs in the TRY block, control is passed to the CATCH block.


```
sql  Copy code  
  
BEGIN TRY UPDATE Employees SET Salary = Salary + 1000 WHERE EmployeeId = 1; END TRY  
BEGIN CATCH PRINT 'Error Occurred: ' + ERROR_MESSAGE(); END CATCH;
```

### THROW vs RAISERROR

- **Question:** What is the difference between THROW and RAISERROR in T-SQL?

**Answer:**

- **RAISERROR:** Used for backward compatibility, allows custom error messages, severity levels, and error numbers.
- **THROW:** Introduced in SQL Server 2012, simpler syntax, and rethrows the original error without changing the error number or message.

```
sql  Copy code  
  
THROW 50001, 'Custom error message', 1;
```


---

## 8. Transactions in T-SQL

### BEGIN TRANSACTION, COMMIT, and ROLLBACK

- **Question:** How do you control transactions in T-SQL?


**Answer:** Transactions are controlled using `BEGIN TRANSACTION`, `COMMIT`, and `ROLLBACK`. `BEGIN TRANSACTION` marks the start of a transaction, `COMMIT` saves the changes, and `ROLLBACK` undoes the changes if an error occurs.

```
sql  Copy code  
  
BEGIN TRANSACTION; UPDATE Employees SET Salary = 50000 WHERE EmployeeId = 10; IF  
@@ERROR <> 0 ROLLBACK; ELSE COMMIT;
```

### Savepoints

- **Question:** What are savepoints in T-SQL, and how are they used?

**Answer:** Savepoints allow you to create intermediate points within a transaction. If an error occurs, you can roll back to a savepoint without undoing the entire transaction.

```
sql  Copy code  
  
BEGIN TRANSACTION; SAVE TRANSACTION SavePoint1; -- Perform some operation IF  
@@ERROR <> 0 ROLLBACK TRANSACTION SavePoint1; -- Rollback to savepoint COMMIT;
```


---

## 9. Temporary Tables and Table Variables

### Temporary Tables

- **Question:** What is a temporary table in T-SQL, and how is it created?


**Answer:** Temporary tables are used to store intermediate results. They are created in the `tempdb` database and are automatically dropped when the session ends. Local temporary tables begin with a `#`.

```
sql  Copy code  
  
CREATE TABLE #TempEmployees (EmployeeId INT, Name NVARCHAR(100)); INSERT INTO  
#TempEmployees VALUES (1, 'John Doe');
```

### Table Variables

- **Question:** How do table variables differ from temporary tables in T-SQL?

**Answer:** Table variables are similar to temporary tables but are scoped to the batch, function, or stored procedure. They are declared using `DECLARE`, and their lifespan is limited to the current session.

```
sql  Copy code  
  
DECLARE @TempTable TABLE (EmployeeId INT, Name NVARCHAR(100)); INSERT INTO  
@TempTable VALUES (1, 'John Doe');
```

---

## 10. Window Functions in T-SQL

### ROW\_NUMBER()

## ROW\_NUMBER()

- **Question:** How does the `ROW_NUMBER()` function work in T-SQL?

**Answer:** `ROW_NUMBER()` assigns a unique, sequential integer to rows within a result set, starting from 1 for the first row. It requires an `OVER()` clause, which defines the partitioning and ordering of the rows.

```
SELECT FirstName, LastName, ROW_NUMBER() OVER (ORDER BY LastName) AS RowNum FROM Employees;
```

## RANK() vs DENSE\_RANK()

- **Question:** What is the difference between `RANK()` and `DENSE_RANK()` in T-SQL?

**Answer:**

- `RANK()` : Assigns a rank to rows, with gaps in rank values when there are ties.
- `DENSE_RANK()` : Assigns ranks without gaps, meaning if two rows tie, the next row gets the immediate next rank.

```
SELECT FirstName, LastName, RANK() OVER (ORDER BY Salary DESC) AS Rank FROM Employees; SELECT FirstName, LastName, DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank FROM Employees;
```

## NTILE()

- **Question:** How does the `NTILE()` function work in T-SQL?

**Answer:** `NTILE()` divides rows into a specified number of groups and assigns a group number to each row. It's useful for dividing data into quartiles, percentiles, or any other equal-sized partitions.

```
SELECT FirstName, LastName, NTILE(4) OVER (ORDER BY Salary DESC) AS Quartile FROM Employees;
```

---

## 11. Pivoting and Unpivoting Data

### PIVOT

- **Question:** How do you use the `PIVOT` operator in T-SQL?

**Answer:** `PIVOT` is used to transform rows into columns. It aggregates data and is helpful for summarizing data in reports.

```
SELECT Department, [2019], [2020], [2021] FROM ( SELECT Department, Year, Revenue
FROM DepartmentRevenue ) AS SourceTable PIVOT ( SUM(Revenue) FOR Year IN ([2019],
[2020], [2021]) ) AS PivotTable;
```

## UNPIVOT

- **Question:** How do you use the `UNPIVOT` operator in T-SQL?

**Answer:** `UNPIVOT` converts columns back into rows, which is the reverse of `PIVOT`. It can be used to normalize denormalized data.

```
SELECT Department, Year, Revenue FROM DepartmentRevenue UNPIVOT ( Revenue FOR Year
IN ([2019], [2020], [2021]) ) AS UnpivotTable;
```

---

## 12. Set Operations

### UNION vs UNION ALL

- **Question:** What is the difference between `UNION` and `UNION ALL` in T-SQL?

**Answer:**

- `UNION` : Combines results of two or more queries and removes duplicates.
- `UNION ALL` : Combines results of two or more queries without removing duplicates (faster than `UNION` because no duplicate checks are performed).

```
SELECT FirstName, LastName FROM Employees_A UNION SELECT FirstName, LastName FROM
Employees_B; SELECT FirstName, LastName FROM Employees_A UNION ALL SELECT
FirstName, LastName FROM Employees_B;
```

### EXCEPT and INTERSECT

- **Question:** How do `EXCEPT` and `INTERSECT` work in T-SQL?

**Answer:**

- `EXCEPT` : Returns the rows from the first query that are not in the second query.
- `INTERSECT` : Returns the rows common to both queries.

```
SELECT FirstName, LastName FROM Employees_A EXCEPT SELECT FirstName, LastName FROM
Employees_B; SELECT FirstName, LastName FROM Employees_A INTERSECT SELECT
FirstName, LastName FROM Employees_B;
```

---

## 13. Advanced Querying Techniques

### Recursive CTEs

- **Question:** How do you implement a recursive CTE in T-SQL, and what are the use cases?

**Answer:** Recursive CTEs are used to perform hierarchical queries, such as retrieving data from a tree or parent-child structure. A recursive CTE consists of two parts: the anchor member and the recursive member.

```
WITH EmployeeHierarchy AS ( SELECT EmployeeId, ManagerId, FirstName, LastName FROM
Employees WHERE ManagerId IS NULL -- Anchor member UNION ALL SELECT E.EmployeeId,
E.ManagerId, E.FirstName, E.LastName FROM Employees E INNER JOIN EmployeeHierarchy
EH ON E.ManagerId = EH.EmployeeId -- Recursive member ) SELECT * FROM
EmployeeHierarchy;
```

#### APPLY (CROSS APPLY and OUTER APPLY)

- **Question:** What are CROSS APPLY and OUTER APPLY, and how do they work in T-SQL?

**Answer:**

- **CROSS APPLY:** Works like an INNER JOIN, applying a table-valued function to each row of the outer table.
- **OUTER APPLY:** Works like a LEFT JOIN, returning all rows from the outer table and matching rows from the function.

```
SELECT E.EmployeeId, E.FirstName, P.ProjectName FROM Employees E CROSS APPLY (
SELECT TOP 1 ProjectName FROM Projects P WHERE P.EmployeeId = E.EmployeeId ORDER BY
StartDate DESC ) AS RecentProject;
```

## 14. Data Integrity and Constraints

### Primary Key and Foreign Key Constraints

- **Question:** What is the difference between primary key and foreign key constraints in T-SQL?

**Answer:**

- **Primary Key:** Uniquely identifies each row in a table. Each table can have only one primary key.
- **Foreign Key:** Establishes a relationship between two tables by linking the foreign key in one table to the primary key in another table, enforcing referential integrity.

```
ALTER TABLE Employees ADD CONSTRAINT PK_EmployeeId PRIMARY KEY (EmployeeId); ALTER
TABLE Projects ADD CONSTRAINT FK_EmployeeId FOREIGN KEY (EmployeeId) REFERENCES
Employees(EmployeeId);
```

### UNIQUE and CHECK Constraints

- **Question:** How do UNIQUE and CHECK constraints work in T-SQL?

**Answer:**

- **UNIQUE Constraint:** Ensures that all values in a column are unique, but unlike primary keys, a table can have multiple unique constraints.
- **CHECK Constraint:** Enforces a condition that must be true for each row in a table.

```
ALTER TABLE Employees ADD CONSTRAINT UQ_Email UNIQUE (Email); ALTER TABLE Employees  
ADD CONSTRAINT CK_Salary CHECK (Salary > 0);
```

---

## 15. T-SQL Best Practices

### Optimizing Query Performance

- **Question:** What are some best practices for optimizing query performance in T-SQL?

**Answer:** Some best practices include:

- **Use proper indexing:** Ensure that frequently queried columns are indexed.
- **Avoid SELECT \*:** Only select the columns you need.
- **Use EXISTS instead of IN:** In certain cases, EXISTS performs better than IN.
- **Minimize joins:** Use joins only when necessary, and avoid joining on columns with no indexes.
- **Monitor query performance:** Use execution plans and query store to analyze and optimize queries.

### Using Transactions Effectively

- **Question:** How can you use transactions effectively in T-SQL?

**Answer:** Best practices for using transactions include:

- **Keep transactions short:** Minimize the number of operations inside a transaction to avoid blocking other queries.
- **Handle errors:** Use TRY...CATCH to handle errors and ensure that transactions are either committed or rolled back properly.
- **Use savepoints:** Create savepoints for large transactions to allow partial rollbacks if necessary.

```
BEGIN TRANSACTION; -- Operations IF @@ERROR <> 0 ROLLBACK TRANSACTION; ELSE COMMIT  
TRANSACTION;
```