**What is RDBMS?**

Relational Database Management Systems (RDBMS) are database management systems that maintain data records and indices in tables. Relationships may be created and maintained across and among the data and tables. In a relational database, relationships between data items are expressed by means of tables. Interdependencies among these tables are expressed by data values rather than by pointers.

This allows for a high degree of data independence. An RDBMS has the capability to recombine the data items from different files, providing powerful tools for data usage.

**Explain DML, DDL, DCL, and TCL statements with examples?**

This is one of the most frequently asked **SQL Server Interview Questions.** Let us understand the above terms in detail.

**DML:**

DML stands for Data Manipulation Language. DML is used to retrieve, insert, update, and delete data in a database that means DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records. DML statements include the following:

**SELECT**– select records from a table  
**INSERT**– insert new records  
**UPDATE**– update/Modify existing records  
**DELETE**– delete existing records

**DDL:**

DDL stands for Data Definition Language. DDL statements are used to alter/modify a database or table structure and schema. These statements handle the design and storage of database objects.

**CREATE**– create a new Table, database, schema  
**ALTER**– alter the existing table, column description  
**DROP**– delete existing objects from a database

**DCL:**

DCL stands for data control language. Generally, we will use these commands to secure database objects by creating roles, permissions using GRANT, REVOKE operations. In SQL Server, the following operations will come under DCL operations

**GRANT**– allows users to read/write on certain database objects  
**REVOKE**– keeps users from the read/write permission on database objects

**TCL:**

TCL stands for Transactional Control Language. TCL is used to manage transactions within a database. Examples: COMMIT, ROLLBACK, Begin Transaction statements

**BEGIN Transaction** – opens a transaction  
**COMMIT Transaction** – commits a transaction  
**ROLLBACK Transaction** – ROLLBACK a transaction in case of any error

**What is the difference between Drop, Delete and Truncate statements in SQL Server?**

This is one of the frequently asked SQL Server Interview Questions. The **Drop, Delete, and Truncate** – All operations can be rolled back.

Delete command removes the rows from a table based on the condition that we provide with a WHERE clause. Truncate will actually remove all the rows from a table and there will be no data in the table after we run the truncate command.

All the statements (**Delete, Truncate, and Drop**) are logged operations but the amount of information that is logged varies. **Delete**statement logs an entry in the transaction log for each deleted row, whereas **Truncate Table**logs only the Page deallocations. **Hence, truncate is a little faster than Delete**.

**DELETE:**

1. The DELETE command is used to remove some or all rows from a table.
2. A WHERE clause can be used with a DELETE command to remove some specific rows from a table.
3. If the WHERE condition is not specified, then all rows are removed.
4. The DELETE operation will cause all DELETE triggers on the table to fire.
5. It does not reset the identity of the column value.
6. It removes rows on a row-by-row basis and hence for each deleted row it records an entry in the transaction logs, thus this is slower than truncate.
7. This is a DML command so it is just used to manipulate or modify the table data and it does not change any property of a table.

**TRUNCATE:**

1. TRUNCATE removes **all rows** from a table, but the table structure and its columns, constraints, indexes, and so on remain.
2. It does not require a **WHERE** clause, so we cannot filter rows while Truncating.
3. **IDENTITY** columns are re-seeded on this operation if no seed was defined then the default value 1 is used.
4. **No Triggers** are fired on this operation because it does not operate on individual rows.
5. TRUNCATE removes the data by deallocating the data pages used to store the table’s data instead of rows and records, and only the page deallocations are recorded in the transaction log thus it is faster than delete.
6. We cannot use TRUNCATE TABLE on a table referenced by a FOREIGN KEY constraint.
7. TRUNCATE is DDL Command

**DROP:**

1. The DROP command removes a table from the database.
2. All the related Data, Indexes, Triggers, Constraints, and Permission specifications for the Table are dropped by this operation.
3. Some objects like Views, Stored Procedures that reference the dropped table are not dropped and must be explicitly dropped.
4. Cannot drop a table that is referenced by any Foreign Key constraint.
5. **No Triggers** are fired on this operation because it does not operate on individual rows.

**Note:**If TRUNCATE is written in Query Editor surrounded by TRANSACTION and if the session is closed, it cannot be rolled back but DELETE can be rolled back.

**What is the Cascading referential integrity constraint?**

This is one of the most frequently asked SQL Server Interview Questions. Cascading referential integrity constraints are foreign key constraints that tell SQL Server to perform certain actions when a user attempts to delete or update a primary key to which an existing foreign keys point.

**SET NULL:**If a delete or update statement affects rows in a foreign key table, those values will be set to NULL when the primary key record is deleted or updated. The foreign key columns affected must allow NULL values.

**CASCADE:**If a delete statement affects one or more rows in a foreign key table, those rows will be deleted when the primary key record is deleted. If an update statement affects rows in the foreign key table, those rows will be updated with the value from the primary key record after it has been updated.

ADVERTISEMENT

**SET DEFAULT:**If a delete or update statement affects rows in a foreign key table, then all rows containing those foreign keys are set to the default value. All foreign key columns in the related table must have default constraints defined on them.

**NO ACTION:**This is the default action. This specifies that if an update or deletes statement affects rows in foreign key tables, then the action will be denied and rolled back. An error message will be raised.

**What is the difference between where clause and having clause in SQL Server?**

This is one of the most frequently asked SQL Server Interview Questions and in almost all interviews this question being asked.

1. WHERE clause cannot be used with aggregate functions whereas the HAVING clause can be used with aggregate functions. This means the WHERE clause is used for filtering individual rows on a table whereas the HAVING clause is used to filter groups.
2. WHERE comes before GROUP BY. This means the WHERE clause filters rows before aggregate calculations are performed. HAVING comes after GROUP BY. This means the HAVING clause filters rows after aggregate calculations are performed. So, from a performance standpoint, HAVING is slower than WHERE and should be avoided when possible.
3. WHERE and HAVING clause can be used together in a SELECT query. In this case WHERE clause is applied first to filter individual rows. The rows are then grouped and aggregate calculations are performed, and then the HAVING clause filters the groups.
4. WHERE clause can be used with – Select, Insert, and Update statements whereas the HAVING clause can only be used with the Select statement.

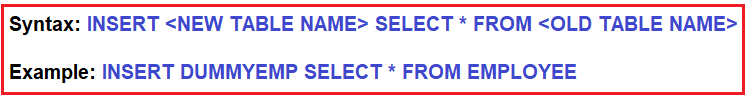
**What are the differences between primary key and unique key in SQL Server?**

This is of the most asked SQL Server Interview Questions in Interviews. Let discuss this question in detail.

1. A table can have only one primary key. On the other hand, a table can have more than one unique key.
2. The primary key column does not accept any null values whereas a unique key column accepts one null value.
3. Both Primary key and unique key enforce the uniqueness of the column on which they are defined. But By default, the primary key creates a unique clustered index on the column whereas a unique key creates a unique non clustered index.

**How can we copy the data from one table to another?**

This is one of the most frequently asked SQL Server Interview Questions and the interviewer basically asked to write the query. When we copy the data from one table to another table then the two tables should contain the same structure.



When we copy the data from one table to another table we use insert and select query. Tables always independent objects that mean a table does not depend on other tables

**How to create a new table from an existing table or in how many ways we can create a new table from an existing table?**

If required we can create a new table from an existing table as below.

**Syntax1:** (with all column from an existing table)  
**SELECT \* INTO <NEW TABLE NAME> FROM <OLD TABLE NAME>**

**Example: SELECT \* INTO NEWEMPLOYEE FROM EMPLOYEE**

When we execute the above query it will create a new table with all records from an existing table.

**Syntax2:** (with specific columns from an existing table)  
**SELECT <REQUIREDCOLUMN> INTO <NEW TABLE NAME> FROM <OLD TABLE NAME>**

**Example:** **SELECT EID, SALARY INTO SPECEMP FROM EMPLOYEE**  
When we execute the above query it will create a new table with the specific column data from an existing table.

**Syntax3:** (creating a new table without data)  
**SELECT \* INTO <NEW TABLE NAME> FROM <OLD TABLE NAME> WHERE 1 = 0**

**Example:** SELECT \* INTO DUMMYEMP FROM EMPLOYEE WHERE 1 = 0  
**OR**  
**SELECT <REQUIRED COLUMNS> INTO <NEW TABLE NAME> FROM <OLD TABLE NAME>**  
**SELECT EID, SALARY INTO TAB1 FROM EMPLOYEE WHERE 1 = 0**

When we execute the above query it will create a new table without records from an existing table.

**What is normalization?**

[**Database normalization**](https://dotnettutorials.net/lesson/database-normalization-sql-server/)is a data design and organization process applied to data structures based on rules that help build relational databases. In relational database design the process of organizing data to minimize redundancy. Normalization usually involves dividing a database into two or more tables and defining relationships between the tables. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

**What are the different normalization forms?**

This is one of the frequently asked SQL Server Interview Questions.

**1NF: Eliminate Repeating Groups:**Make a separate table for each set of related attributes, and give each table a primary key. Each field contains at most one value from its attribute domain.

**2NF: Eliminate Redundant Data:**If an attribute depends on only part of a multi-valued key, remove it to a separate table.

**3NF: Eliminate Columns Not Dependent On Key:**If attributes do not contribute to a description of the key, remove them to a separate table. All attributes must be directly dependent on the primary key

**BCNF: Boyce-Codd Normal Form:**If there are non-trivial dependencies between candidate key attributes, separate them out into distinct tables.

**4NF: Isolate Independent Multiple Relationships:**No table may contain two or more 1:n or n:m relationships that are not directly related.

**5NF: Isolate Semantically Related Multiple Relationships:**There may be practical constrains on information that justifies separating logically related many-to-many relationships.

**ONF: Optimal Normal Form:**A model limited to only simple (elemental) facts, as expressed in Object Role Model notation.

**DKNF: Domain-Key Normal Form:**A model free from all modification anomalies.

Remember, these normalization guidelines are cumulative. For a database to be in 3NF, it must first fulfill all the criteria of a 2NF and 1NF database. Please [**click here**](https://dotnettutorials.net/lesson/database-normalization-sql-server/) to learn [**Database Normalization**](https://dotnettutorials.net/lesson/database-normalization-sql-server/) in detail step by step with some real-time examples.

**What is the Cursor?**

A cursor is a database object used by applications to manipulate data in a set on a row-by-row basis, instead of the typical SQL commands that operate on all the rows in the set at one time.

In order to work with a cursor we need to perform some steps in the following order:

1. Declare cursor
2. Open cursor
3. Fetch row from the cursor Process fetched row Close cursor
4. Deallocate cursor

**What is the use of DBCC commands?**

DBCC stands for database consistency checker. We use these commands to check the consistency of the databases, i.e., maintenance, validation task, and status checks. For example,

1. **DBCC CHECKDB** – Ensures that tables in the DB and the indexes are correctly linked.
2. **DBCC CHECKALLOC** – To check that all pages in a DB are correctly allocated.
3. **DBCC CHECKFILEGROUP** – Checks all tables file group for any damage.

**What is a Linked Server?**

Linked Servers is a concept in SQL Server by which we can add other SQL Server to a Group and query both the SQL Server DBS using T-SQL Statements. With a linked server, you can create very clean, easy to follow, SQL statements that allow remote data to be retrieved, joined, and combined with local data. Stored Procedure **sp\_addlinkedserver**, **sp\_addlinkedsrvlogin** will be used to add a new Linked Server.

**What is Collation?**

Collation refers to a set of rules that determine how data is sorted and compared. Character data is sorted using rules that define the correct character sequence, with options for specifying case-sensitivity, accent marks, kana character types, and character width.

**What are the different types of Collation Sensitivity?**

1. **Case sensitivity:**A and a, B and b, etc.
2. **Accent sensitivity:** a and á, o and ó, etc.
3. **Kana Sensitivity:** When Japanese kana characters Hiragana and Katakana are treated differently, it is called Kana sensitive.
4. **Width sensitivity:**When a single-byte character (half-width) and the same character when represented as a double-byte character (full-width) are treated differently than it is width sensitive.

**How to implement one-to-one, one-to-many and many-to-many relationships while designing tables?**

1. The one-to-one relationship can be implemented as a single table and rarely as two tables with primary and foreign key relationships.
2. One-to-Many relationships are implemented by splitting the data into two tables with primary key and foreign key relationships.
3. Many-to-Many relationships are implemented using a junction table with the keys from both the tables forming the composite primary key of the junction table.

**What is a NOLOCK?**

This is one of the most frequently asked SQL Interview Questions. Using the NOLOCK query optimizer hint is generally considered good practice in order to improve concurrency on a busy system. When the NOLOCK hint is included in a SELECT statement, no locks are taken when data is read. The result is a Dirty Read, which means that another process could be updating the data at the exact time you are reading it. There are no guarantees that your query will retrieve the most recent data.

The advantage of performance is that your reading of data will not block updates from taking place, and updates will not block your reading of data. SELECT statements take Shared (Read) locks. This means that multiple SELECT statements are allowed simultaneous access, but other processes are blocked from modifying the data. The updates will queue until all the reads have completed, and reads requested after the update will wait for the updates to complete. The result of your system is the delay(blocking).

**When is the use of the UPDATE\_STATISTICS command?**

This is one of the most frequently asked SQL Interview Questions. This command is basically used when a large processing of data has occurred. If a large number of deletions any modification or Bulk Copy into the tables has occurred, it has to update the indexes to take these changes into account. UPDATE\_STATISTICS updates the indexes on these tables accordingly.

**What is sub-query? Explain the properties of sub-query.**

This is one of the frequently asked SQL Server Interview Questions. Sub-queries are often referred to as sub-selects, as they allow a SELECT statement to be executed arbitrarily within the body of another SQL statement. A sub-query is executed by enclosing it in a set of parentheses. Sub-queries are generally used to return a single row as an atomic value, though they may be used to compare values against multiple rows with the IN keyword.

A subquery is a SELECT statement that is nested within another T-SQL statement. A subquery SELECT statement if executed independently of the T-SQL statement, in which it is nested, will return a result set. Meaning a subquery SELECT statement can stand alone and is not dependent on the statement in which it is nested. A subquery SELECT statement can return any number of values and can be found in, the column list of a SELECT statement, a FROM, GROUP BY, HAVING, and/or ORDER BY clauses of a T-SQL statement. A Subquery can also be used as a parameter to a function call. Basically, a subquery can be used anywhere an expression can be used.

**Properties of Sub-Query**

1. A subquery must be enclosed in the parenthesis.
2. The subquery must be put in the right hand of the comparison operator, and
3. The subquery cannot contain an ORDER-BY clause.
4. A query can contain more than one sub-queries.

**What are the types of sub-queries?**

1. Single-row subquery, where the subquery returns only one row.
2. Multiple-row subquery, where the subquery returns multiple rows
3. Multiple column subquery, where the subquery returns multiple columns.

**What is the SQL Profiler?**

This is one of the most frequently asked SQL Server Interview Questions. SQL Profiler is a graphical tool that allows system administrators to monitor events in an instance of Microsoft SQL Server. You can capture and save data about each event to a file or SQL Server table to analyze later. For example, you can monitor a production environment to see which stored procedures are hampering performance by executing too slowly.

Use SQL Profiler to monitor only the events in which you are interested. If traces are becoming too large, you can filter them based on the information you want, so that only a subset of the event data is collected. Monitoring too many events adds overhead to the server and the monitoring process and can cause the trace file or trace table to grow very large, especially when the monitoring process takes place over a long period of time.

**Which TCP/IP port does SQL Server run on? How can it be changed?**

SQL Server runs on port 1433. It can be changed from the Network Utility TCP/IP properties –> Port number. both on the client and the server.

**What are the authentication modes in SQL Server? How can it be changed?**

Windows mode and mixed mode (SQL & Windows). To change authentication mode in SQL Server click Start, Programs, Microsoft SQL Server, and click SQL Enterprise Manager to run SQL Enterprise Manager from the Microsoft SQL Server program group. Select the server then from the Tools menu select SQL Server Configuration Properties, and choose the Security page.

**Where are SQL server users names and passwords are stored in SQL server?**

They get stored in master DB in the sysxlogins table.

**Which command using Query Analyzer will give you the version of SQL server and operating system?**

SELECT SERVERPROPERTY(‘productversion’), SERVERPROPERTY (‘productlevel’), SERVERPROPERTY (‘edition’)

**What is the SQL server agent?**

This is one of the frequently asked SQL Server Interview Questions. SQL Server agent plays an important role in the day-to-day tasks of a database administrator (DBA). It is often overlooked as one of the main tools for SQL Server management. Its purpose is to ease the implementation of tasks for the DBA, with its full-function scheduling engine, which allows you to schedule your own jobs and scripts.

**What is log shipping?**

Log shipping is the process of automating the backup of database and transaction log files on a production SQL server and then restoring them onto a standby server. Enterprise Editions only supports log shipping. In log shipping, the transnational log file from one server is automatically updated into the backup database on the other server. If one server fails, the other server will have the same DB that can be used as the Disaster Recovery plan. The key feature of log shipping is that it will automatically backup transaction logs throughout the day and automatically restore them on the standby server at a defined interval.

**What command do we use to rename a DB?**

sp\_renamedb ‘oldname’, ‘newname’

If someone is using DB it will not accept sp\_renmaedb. In that case, first, bring DB to the single user using sp\_dboptions. Use sp\_renamedb to rename the database. Use sp\_dboptions to bring the database to multi-user mode.

**What are sp\_configure commands and set commands?**

Use sp\_configure to display or change server-level settings. To change database-level settings, use ALTER DATABASE. To change settings that affect only the current user session, use the SET statement.

**What are the different types of replication? Explain.**

The SQL Server 2000-supported replication types are as follows:

1. Transactional
2. Snapshot
3. Merge

Snapshot replication distributes data exactly as it appears at a specific moment in time and does not monitor for updates to the data. Snapshot replication is best used as a method for replicating data that changes infrequently or where the most up-to-date values (low latency) are not a requirement. When synchronization occurs, the entire snapshot is generated and sent to Subscribers.

In transactional replication, an initial snapshot of data is applied at Subscribers, and then when data modifications are made at the Publisher, the individual transactions are captured and propagated to Subscribers.

Merge replication is the process of distributing data from Publisher to Subscribers, allowing the Publisher and Subscribers to make updates while connected or disconnected, and then merging the updates between sites when they are connected.

**What are the OS services that the SQL Server installation adds?**

MS SQL SERVER SERVICE, SQL AGENT SERVICE, DTC (Distribution transac co-ordinator)

**What are three SQL keywords used to change or set someone’s permissions?**

GRANT, DENY, and REVOKE.

**What does it mean to have quoted\_identifier on? What are the implications of having it off?**

When SET QUOTED\_IDENTIFIER is ON, identifiers can be delimited by double quotation marks, and literals must be delimited by single quotation marks. When SET QUOTED\_IDENTIFIER is OFF, identifiers cannot be quoted and must follow all Transact-SQL rules for identifiers.

**What is the STUFF function and how does it differ from the REPLACE function?**

STUFF function to overwrite existing characters. Using this syntax, STUFF(string\_expression, start, length, replacement\_characters), string\_expression is the string that will have characters substituted, the start is the starting position, the length is the number of characters in the string that are substituted, and replacement\_characters are the new characters interjected into the string.

REPLACE function to replace existing characters of all occurrences. Using this syntax REPLACE(string\_expression, search\_string, replacement\_string), where every incidence of search\_string found in the string\_expression will be replaced with replacement\_string. Using query analyzer, name 3 ways to get an accurate count of the number of records in a table?

**SELECT \* FROM table1 SELECT COUNT(\*) FROM table1**

**SELECT rows FROM sysindexes WHERE id = OBJECT\_ID(table1) AND indid < 2**

**How to rebuild the Master Database?**

This is one of the frequently asked SQL Server Questions. Shutdown Microsoft SQL Server 2000, and then run Rebuildm.exe. This is located in the Program Files\Microsoft SQL Server\80\Tools\Binn directory.

In the Rebuild Master dialog box, click Browse. In the Browse for Folder dialog box, select the \Data folder on the SQL Server 2000 compact disc or in the shared network directory from which SQL Server 2000 was installed, and then click OK.

Click Settings. In the Collation Settings dialog box, verify or change settings used for the master database and all other databases.

Initially, the default collation settings are shown, but these may not match the collation selected during setup. You can select the same settings used during setup or select new collation settings. When done, click OK.

In the Rebuild Master dialog box, click Rebuild to start the process. The Rebuild Master utility reinstalls the master database. To continue, you may need to stop a server that is running.

**What are the basic functions for master, MSDB, model, TEMPFB databases?**

This is one of the frequently asked SQL Server Interview Questions. The **Master** database holds information for all databases located on the SQL Server instance and is the glue that holds the engine together. Because SQL Server cannot start without a functioning master database, you must administer this database with care.

The **MSDB**database stores information regarding database backups, SQL Agent information, DTS packages, SQL Server jobs, and some replication information such as for log shipping.

The **TEMPDB**holds temporary objects such as global and local temporary tables and stored procedures. The **model** is essentially a template database used in the creation of any new user database created in the instance.

**What is De-normalization?**

[**De-normalization**](https://dotnettutorials.net/lesson/de-normalization-in-sql-server/) is the process of attempting to optimize the performance of a database by adding redundant data. It is sometimes necessary because current DBMSs implement the relational model poorly. A true relational DBMS would allow for a fully normalized database at the logical level while providing physical storage of data that is tuned for high performance. [**De-normalisation**](https://dotnettutorials.net/lesson/de-normalization-in-sql-server/)is a technique to move from higher to lower normal forms of database modeling in order to speed up database access.

**What is Scheduled Jobs or What is a Scheduled Tasks?**

This is one of the frequently asked SQL Server Interview Questions. Scheduled tasks let users automate processes that run on regular or predictable cycles. Users can schedule administrative tasks, such as cube processing, to run during times of slow business activity. Users can also determine the order in which tasks run by creating job steps within a SQL Server Agent job. E.g. Back up the database, Update the Stats of Tables. Job steps give the user control over the flow of execution.

If one job fails, the user can configure SQL Server Agent to continue to run the remaining tasks or to stop the execution.

**What is BCP? When does it use?**

BulkCopy is a tool used to copy a huge amount of data from tables and views. BCP does not copy the structures same as source to destination.

**How do you load large data to the SQL server database?**

BulkCopy is a tool used to copy a huge amount of data from tables. BULK INSERT command helps to Imports a data file into a database table or view in a user-specified format.

**Can we rewrite subqueries into simple select statements or with joins?**

Subqueries can often be re-written to use a standard outer join, resulting in faster performance. As we may know, an outer join uses the plus sign (+) operator to tell the database to return all non-matching rows with NULL values. Hence we combine the outer join with a NULL test in the WHERE clause to reproduce the result set without using a sub-query.

**Can SQL Servers link to other servers like Oracle?**

This is one of the most frequently asked SQL Server Interview Questions. SQL Server can be lined to any server provided it has an OLE-DB provider from Microsoft to allow a link. E.g. Oracle has an OLE-DB provider for oracle that Microsoft provides to add it as a linked server to the SQL Server group.

**How to copy the tables, schema, and views from one SQL server to another?**

Microsoft SQL Server 2000 Data Transformation Services (DTS) is a set of graphical tools and programmable objects that lets user extract, transform, and consolidate data from disparate sources into single or multiple destinations.

**What is Data Warehousing?**

**Subject-oriented**, meaning that the data in the database is organized so that all the dataelements relating to the same real-world event or object are linked together;

**Time-variant**, meaning that the changes to the data in the database are tracked and recordedso that reports can be produced showing changes over time;

**Non-volatile**, meaning that data in the database is never over-written or deleted, oncecommitted, the data is static, read-only, but retained for future reporting;

**Integrated**, meaning that the database contains data from most or all of an organization’soperational applications and that this data is made consistent.

**What is OLTP (Online Transaction Processing)?**

In OLTP – online transaction processing systems relational database design uses the discipline of data modeling and generally follows the Codd rules of data normalization in order to ensure absolute data integrity. Using these rules complex information is broken down into its most simple structures (a table) where all of the individual atomic level elements relate to each other and satisfy the normalization rules.

**What is Data Integrity?**

1. Data integrity means the data contained in the database is accurate and reliable.
2. To provide data integrity, RDBMS provides a set of integrity constraints that ensures that data entered into the database is accurate, valid, and consistent.

**What are the different levels of data integrity in SQL Server?**

1. Entity Integrity (uniquely identify- PK, UQ)
2. Domain Integrity (Follow defined rule- Check
3. Referential integrity (relationships- FK)
4. User-defined integrity

**What is Entity integrity?**

1. Entity integrity ensures each row in a table is a uniquely identifiable entity. We can achieve entity integrity in a table by using either PRIMARY KEY or UNIQUE KEY constraints.
2. In simple words, we can say Entity Integrity ensures that there are no duplicate rows in a table.

**What is Referential integrity?**

1. Referential integrity ensures that relationships between tables remain preserved as data is inserted, deleted, and modified. We can apply referential integrity using a FOREIGN KEY constraint.
2. In simple words, we can say that Referential integrity ensures that rows cannot be updated or deleted which are used by other records.

**What is Domain Integrity?**

1. Domain integrity ensures that the values going to store in a column must follow some defined rules such as range, type, and format. A database can enforce these rules by using the CHECK KEY constraint.
2. In simple words, we can say that Domain Integrity enforces valid entries for a given column by restricting the type, the format, or the range of possible values.

**What is user-defined integrity?**

User-Defined Integrity enforces some specific business rules that do not fall into an entity, domain, or referential integrity categories. Each of these categories of data integrity can be enforced by the appropriate constraints.

**What is a Constraint in SQL Server?**

A [**constraint**](https://dotnettutorials.net/lesson/sql-server-constraints/)is a property that is assigned to a column or set of columns in a table that is used to enforce data integrity means it ensures that the data is going to store in a table is valid, consistent, and accurate.

**Why do we need Constraints in SQL Server?**

1. A [**constraint**](https://dotnettutorials.net/lesson/sql-server-constraints/)is used to restrict the insertion of unwanted data in any columns.
2. We can create constraints on single or multiple columns of any table.
3. It maintains the data integrity i.e. accurate data or original data of the table.

**What are the different types of Constraints available in SQL Server?**

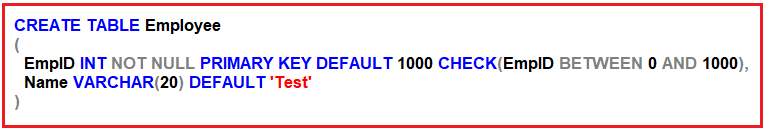
SQL Server supports five constraints for maintaining data integrity which are

1. [**UNIQUE KEY constraint**](https://dotnettutorials.net/lesson/sql-server-constraints/)
2. [**NOT NULL constraint**](https://dotnettutorials.net/lesson/sql-server-constraints/)
3. [**CHECK KEY constraint**](https://dotnettutorials.net/lesson/sql-server-constraints/)
4. [**PRIMARY KEY constraint**](https://dotnettutorials.net/lesson/primary-key-constraint-sql-server/)
5. [**FOREIGN KEY constraint.**](https://dotnettutorials.net/lesson/foreign-key-constraint-sql-server/)

**Note:** Constraints are imposed on columns of a table.

**What is Default Constraint in SQL Server?**

1. The default constraint is used to fill the column with a default value that is defined during the creation of a table if the user does not supply any value while inserting the data.
2. IDENTITY columns and timestamp columns can’t be associated with a default constraint.
3. In simple words, we can say that Default constraints enable the SQL Server to write default value to a column when the user doesn’t specify a value.
4. A default allows us to specify a constant value, NULL, or the run-time value of a system function if no known value exists or if the column is missing in an INSERT statement.
5. Let us see an example for understanding this concept



**What is NOT NULL Constraint in SQL Server?**

1. If a NOT NULL constraint is imposed on a column then that column will not allow null values into it.
2. The NOT NULL Constraint is used to avoid NULL values but accepted duplicate values into a column.
3. It enforces that the column will not accept NULL values. The NOT NULL constraints are used to enforce domain integrity as the check constraints.
4. A table can contain any number of NOT NULL constraints.
5. We can apply the NOT NULL constraint on any data type column such as integer, character, money, etc.

Note: When we INSERT a null value into a column on which the NOT NULL constraint is imposed. The execution of the insert statement is terminated by displaying a user-friendly message telling the reason for termination and also specifies the database, the table, and the column where the problem got occurred.

**What is a Unique Constraint in SQL Server?**

1. If this constraint is imposed on a column then that column will not allow duplicate values into it.
2. A UNIQUE constraint is used to avoid duplicate values but accepted null values in a column.
3. IT enforces the uniqueness of the values in a set of columns, so no duplicate values are entered. The unique key constraints are used to enforce entity integrity as the primary key constraints.
4. A table can contain any number of UNIQUE constraints.
5. We can apply the UNIQUE constraint on any data type column such as integer, character, money, etc.
6. The UNIQUE constraint will accept 1 NULL value only.

**Note:**The unique key constraint is used to make sure that there is no duplicate value in that column. Both unique key and primary key enforces the uniqueness of the column but there is one difference between them unique key constraint allows null value but the primary key does not allow null value.

**IMPOSING CONSTRAINT ON TABLE:**

We can impose constraints on a table in two different ways

1. Column-level imposing a constraint
2. Table level imposing a constraint

In the first case, we provide the constraint information inside the column only whereas in the second case we first define all the columns and then we define constraints on the columns.

**Note:** We cannot impose NOT NULL constraint in table level. It is possible only for the rest of the four constraints.

**Column Level Imposing a Constraint:**

CREATE TABLE CUSTOMER

**(**

CID INT CONSTRAINT CUSTID\_UNIQUE UNIQUE,

CNAME VARCHAR**(**50**)**,

BALANCE MONEY,

CITY VARCHAR**(**50**)**

**)**

**Table Level Imposing a Constraint:**

CREATE TABLE CUSTOMER

**(**

CUSTID INT,

CNAME VARCHAR**(**50**)**,

BALANCE MONEY,

CITY VARCHAR**(**50**)**,

CONSTRAINT CUSTID\_UNIQUE UNIQUE**(**CUSTID**)**

**)**

**What is a composite constraint in SQL Server?**

There is no difference in behavior whether the constraint is imposed at table level or column level but if a constraint is imposed at table level we have the advantage of imposing composite constraints. That is one constraint on multiple columns. **For example:**

CREATE TABLE BRANCHDETAILS

**(**

CITY VARCHAR**(**50**)**,

BRANCHCODE VARCHAR**(**10**)**,

BRANCHLOCATION VARCHAR **(**30**)**,

CONSTRAINT CITY\_BC\_UNIQUE UNIQUE**(**CITY, BRANCHCODE**)**

**)**

**Note:** The drawback with a NOT NULL constraint is it will allow duplicate values whereas in the case of the UNIQUE constraint it will allow null values.

**What is CHECK Constraint in SQL Server?**

1. The CHECK constraint is used to enforce domain integrity.
2. Domain integrity ensures that the values going to store in a column must follow some defined rules such as range, type, and format.
3. In simple words, we can say that Domain Integrity enforces valid entries for a given column by restricting the type, the format, or the range of possible values.
4. Check constraints allow us to define an expression for a TABLE that must not evaluate to FALSE for a data modification statement to succeed.

**Example:**

CREATE TABLE EMPLOYEE

**(**

EMP\_ID INT NOT **NULL** PRIMARY KEY CHECK**(**EMP\_ID BETWEEN 0 AND 1000**)**,

EMP\_NAME VARCHAR**(**30**)** NOT **NULL**,

ENTERED\_DATE DATETIME NOT **NULL** CHECK **(**ENTERED\_DATE **>**= CURRENT\_TIMESTAMP**)**,

DEPT\_NO INT CHECK**(**DEPT\_NO **>** 0 AND DEPT\_NO **<** 100**)**

**)**

Check constraints will be useful to limit the range of possible values in a column.

**We could create check constraints at two different levels**

1. Column-level check constraints are applied only to the column and cannot reference data in other columns
2. Table-level check constraints can reference any column within a table but cannot reference columns in other tables

A table can contain any number of check constraints and will apply to any column data type like integer, character, and decimal, date, etc.

**What is the Primary Key Constraint in SQL Server?**

1. PRIMARY KEY is the combination of UNIQUE and NOT NULL constraint which does not allow either NULL or Duplicate values into a column or columns.
2. Using the primary key we can enforce entity integrity.
3. PRIMARY KEY is also used to make a relationship with the FOREIGN KEY constraint on the table.
4. A table should contain 1 PRIMARY KEY constraint only which can be either on a single or multiple columns i.e. composite primary key also allowed.
5. Every TABLE should have a primary key constraint to uniquely identify each row and only one primary key constraint can be created for each table.
6. PRIMARY KEY can apply to any data type like integer, character, decimal, money, etc.

**What is a Composite Primary Key in SQL Server?**

1. When the primary key constructed with more than 1 column is called a composite primary key.
2. The maximum number of columns is including in the composite primary key is 16 columns.
3. The composite primary key we need to define after all columns definition i.e. at the end of the table definition.
4. In a composite primary key, each column can accept duplicate values but the duplicate combinations should not be duplicated.

**Note:** The primary key is also called a candidate key.

**What is Foreign Key Constraint in SQL Server?**

1. One of the most important concepts in a database is creating a relationship between database tables.
2. These relationships provide a mechanism for linking data stores in multiple tables and retrieving them in an efficient manner.
3. In order to create a link between two tables, we must specify a FOREIGN KEY in one table that references a column in another table.
4. FOREIGN KEY constraints are used for relating or binding two tables with each other and then verify the existence of one table data in other tables.
5. A foreign key in one TABLE points to a primary key in another table. Foreign keys prevent actions that would leave rows with foreign key values when there are no primary keys with that value. Foreign key constraints are used to enforce referential integrity.

**To impose a FOREIGN KEY Constraint we require the following things**

1. We require two tables for binding with each other and those two tables must have a common column for linking the tables.
2. The common column that is present in both the tables need not have the same name but their data type must be the same.
3. The common column that is present under the parent table or master table is known as a reference key column and the reference key column should not contain duplicate values in it. So we need to impose either UNIQUE or PRIMARY key constraint on that column.
4. If we impose a primary key constraint on the reference key column that column will also be the identity column of the table.
5. The common column which is present in the child or detailed table is known as the FOREIGN key column and we need to impose a FOREIGN key constraint on the column which refers to the reference key column of the master table.

**How can we provide the default value for a column?**

The default, default value for any column is NULL, provided the column is not imposed with a NOT NULL constraint. At the time of creating the table, we have a chance of giving our own default value on the column. So that when we insert a record into the table without specifying a value to that column automatically the default value comes into the picture and inserted into the column whereas if we provide a value that provided only will be inserted.

**What is SELF REFERENTIAL INTEGRITY?**

This is the same as the referential integrity we have learned earlier. In earlier cases, we are binding one column of a table with another column of another table whereas in self-referential integrity we bind a column of a table with another column of the same table i.e. both the foreign key and primary key will be present in one table only.

**Steps to view the primary key and foreign key relationship ER diagram representation.**

1. Go to open SQL Server management studio
2. Click on the new query option
3. Go to the query option from the main menu bar
4. Click on design query in the editor option
5. Select primary key and foreign key tables one by one and click on the add button
6. Click on the close button

**What is the difference between a primary key and a unique key?**

This is one of the frequently asked SQL Server Interview Questions. Let us understand the difference between them.

Both primary key and unique key enforces the uniqueness of the column on which they are defined. But by default primary key creates a unique clustered index on the column where are unique key creates a unique non-clustered index by default. Another major difference is that the primary key doesn’t allow NULLs, but the unique key allows one NULL only.

**What is the difference between primary key and foreign key?**

This is one of the frequently asked SQL Server Interview Questions. Let us understand the difference between a primary key and a foreign key.

**Primary key:**

1. A primary key uniquely identifies a record in the table.
2. Primary Key can’t accept null values and duplicate values by default,
3. The primary key is a clustered index and data in the database table is physically organized in the sequence of the clustered index.
4. We can have only one Primary key in a table.

**Foreign key:**

1. A foreign key is a field in the table that is the primary key (or unique key) in another table.
2. The foreign key can accept null values and duplicate values.
3. A foreign key does not automatically create an index, clustered, or non-clustered. We can manually create an index on the foreign key.
4. We can have more than one foreign key in a table.

**Note:**We can’t insert a foreign key column value into the table if the primary key value not available but the reverse is possible and we can’t delete the primary key value if the foreign key reference is set into the table but the reverse is possible.

**CAN A TABLE HAVE MULTIPLE UNIQUE, FOREIGN, AND/OR PRIMARY KEYS?**

A table can have multiple unique and foreign keys. However, a table can have only one primary key.

**CAN A UNIQUE KEY HAVE NULL VALUES? CAN A PRIMARY KEY HAVE NULL VALUES?**

Unique key columns are allowed to hold a single NULL value. The values in a primary key column, however, can **never** be NULL.

**CAN A FOREIGN KEY REFERENCE A NON-PRIMARY KEY?**

Yes, a foreign key can actually reference a key that is not the primary key of a table. But a foreign key must reference a unique key.

**CAN A FOREIGN KEY CONTAIN NULL VALUES?**

Yes, a foreign key can hold NULL values. Because foreign keys can reference unique, non-primary keys – which can hold NULL values – this means that foreign keys can themselves hold NULL values as well.

# **SQL Server Temporary Tables Interview Questions**

**What are Temporary tables?**

SQL Server provides the concept of the **temporary table** which helps us in a great way. These tables can be created at **runtime** and can do all kinds of operations that one normal table can do. But, based on the table types, the scope is limited.

Permanent tables get created in the database we specify and remain in the database permanently until we delete (drop) them. On the other hand, temporary tables get created in the TempDB database and are automatically deleted, when they are no longer used.

**What are the 2 types of Temporary Tables in SQL Server?**

1. Local Temporary Tables
2. Global Temporary Tables

**What is a Local Temporary Table?**

the Local temp tables are only available to the current connection for the user, and they are automatically deleted when the user disconnects from instances. The local temporary table name is stared with hash (“#”) sign.

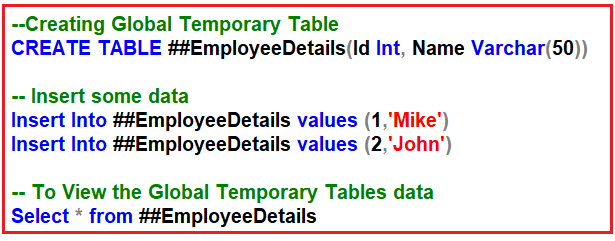
Creating a Local Temporary table is very similar to creating a permanent table, except that we prefix the **table name with 1 pound (#) symbol**. In the example below **#PersonDetails** is a local temporary table, with Id and Name columns.

**For example:**  
**CREATE TABLE #PersonDetails(Id Int, Name Varchar(50))**  
Inserting values into Local Temporary Tables  
**Insert into #PersonDetails values (1,’Mike’)**  
**Insert into #PersonDetails values (2,’John’)**

View the Tables data  
**Select \* from #PersonDetails**

**What are Global Temporary Tables in SQL Server?**

Global Temporary tables name starts with a double hash (“##”). Once this table has been created by a connection, like a permanent table it is then available to any user by any connection. It can only be deleted once all connections have been closed. To create a Global temporary table prefix the name of the table with 2 pounds (##) symbols. For example.



The Global Temporary Tables are visible to all the connections of the SQL Server and are only destroyed when the last connection referencing the table is closed.

Multiple users across multiple connections can have Local temporary tables with the same name but Global Temporary Table Names should be unique and if we inspect the name of the Global Temporary Table in the object explorer there will be no random numbers suffixed at the end of the table name.

**What are the differences between Local and Global Temporary Tables?**

1. Local Temp tables are prefixed with a single pound (#) symbol, whereas global temp tables are prefixed with 2 pounds (##) symbols.
2. SQL Server appends some random numbers at the end of the local temp table name where this is not done for global temp table names.
3. The Local temporary tables are only visible to that session of the Server which has created it, whereas Global temporary tables are visible to all the server sessions.
4. Local temporary tables are automatically dropped; when the session that created, the temporary tables are closed. Global temporary tables are also automatically dropped when the session that creates the temporary tables is closed.

**Can you please explain when to use a temp table in SQL Server?**

When we want to perform many operations on the data in a specific table in the database, we can load the data into a temporary table and then we can perform operations on the temporary table. Loading data into a temporary table speeds up the process because all the operations are performed locally on the client. Use of the Temporary Tables reduces the load on both the network and server as all the interaction with temporary tables occurs on the Client.

1. When we are doing a large number of row manipulation in stored procedures.
2. This is useful to replace the cursor. We can store the result set data into a temp table, then we can manipulate the data from there.
3. When we are having a complex join operation.
4. A Temporary Table variable can be very useful when used with stored procedures to pass input/output parameters or to store the result of a table-valued function.

**Can you create foreign key constraints on temporary tables?**

No

**Do you have to manually delete temporary tables?**

No, temporary tables are automatically dropped, when the session that created the temporary tables is closed. But if we maintain a persistent connection or if connection pooling is enabled, then it is better to explicitly drop the temporary tables we have created. However, it is generally considered a good coding practice to explicitly drop every temporary table we create.

**In which database, the temporary tables get created?**

TEMPDB database.

**How can I check for the existence of a temporary table?**

IF OBJECT\_ID**(**'tempdb..#LocalTempTable'**)** IS NOT **NULL**

Begin

Drop table #LocalTempTable

Print 'Temporary table #LocalTempTable **is** deleted'

End

Else

Begin

Print 'Temporary table #LocalTempTable **is** not Found,

it may have been deleted already'

End

Usually, it is best to create a temp table through the create function. For example, if we want to check if that temp table existed and then drop/create a new one and insert data it would go something like this.

IF object\_ID**(**'tempdb..##temptable'**)** IS NOT **NULL**

DROP TABLE ##TEMPTABLE

CREATE TABLE ##TEMPTABLE

INSERT INTO ##TEMPTABLE

SELECT e.EMPLOYEE

FROM Employee e

WHERE e.employeename = 'Pranaya'

This would check for the table and drop it if it exists then would create a new global temp table (indicated by the ##) and insert all employees with the name Pranaya into this temp table.

**SQL appends random function as at the end of the Local table’s name. So, we can create a Local table with the same name. But the Global table’s name has to be unique. Why is that?**

It is because a Local table is available to the connection that created it. Other users can have that name for their local table. But in the case of a global temporary table, it is available to all users on different sessions. So, its name cannot be different.

**How to copy the structure of the temporary table from one table to another? i.e., copying the only structure and no data?**

Select \* **into** #Temp1 **from** tblEmployee

Select Top 0 \* **into** #Temp2 **from** #Temp1

--Or

Select \* **into** #Temp1 **from** tblEmployee **where** 1=2

**What is the difference between a Temporary Table and a Table variable?**

1. A table variable is created in the memory whereas a temporary table is created in the TempDB. But if there is memory pressure, the pages belonging to a table variable may be pushed out to tempdb.
2. Table variables cannot be involved in transactions, logging or locking. This makes the table variable faster than a temporary table.
3. We can pass the table variable as a parameter to functions and stored procedures, whereas we cannot do the same with a temporary table.
4. The temporary table can have indexes, whereas a table variable can only have a primary index. If speed is an issue Table variable can be faster, but if there are a lot of records, or there is a need to search the temporary table based on a clustered index, then a Temporary Table would be better. If we have less than 100 rows generally use a table variable. Otherwise, use a temporary table. This is because SQL Server won’t create statistics on table variables.

Both **Temporary Tables (#Tables)** and **Table Variables (@Tables)** in SQL Server provide a mechanism for Temporary holding/storage of the result-set for further processing

# **SQL Server Indexes Interview Questions and Answers**

**How is data stored in a database?**

1. SQL Server stores data in it under data pages where a data page is a memory location for storing the information.
2. A data page will be having a size of 8KB and every 8 data pages we stored under a logical container known as “extend”.

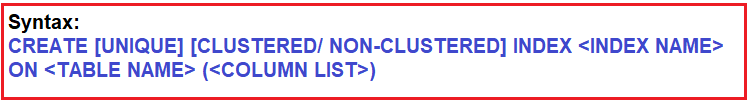
**How will be the database engine retrieves the information from a table?**

This is one of the frequently asked SQL Server Indexes Interview Questions. Whenever the database engine wants to retrieve the information from the table it will adopt two different mechanisms for searching the data

1. **Full page scan**
2. **Index Scan**

In the first case, SQL Server will search for the required information in each and every data page to collect the information. So, if the tables having more number of rows it will take lots of time for searching all the data so it is a time-consuming process.

In the second case SQL Server without searching into each and every data page for retrieving the information it will make use of an index for retrieving the information, where an index is a pointer to the information what we retrieve which can reduce the disk I/O operation saving the time, but if we want to use index scan for searching the data first the index has to be created.



**Note:** Whenever an index is created on a column or columns of a table internally an index table gets created maintaining the information of a column on which the index is created as well as the address (pointer to the row corresponding to a column).

**What is the use of an Index in SQL Server?**

Relational databases like SQL Server use indexes to find data quickly when a query is processed. Creating the proper index can drastically increase the performance of an application.

**What is an index?**

1. An [**index**](https://dotnettutorials.net/lesson/sql-server-indexes/)is a database object which is used by the SQL server to speed up the performance of queries by providing query access to rows in the data table.
2. By using indexes, we can save time and we can improve the performance of database queries and applications.
3. When we create an index on any column SQL Server internally maintains a separate table called index table so that when any user trying to retrieve the data from the existing table depends on index table SQL Server directly go to the table and retrieve required data very quickly.
4. In a table, we can use a maximum of 250 indexes. The index type refers to the way the index is stored internally by SQL Server.

**Why do we need**[**Indexes in SQL Server**](https://dotnettutorials.net/lesson/sql-server-indexes/)**?**

1. Indexes are used by queries to find data from tables quickly. Indexes are created on tables and views. Index on a table or a view is very similar to an index that we find in a book.
2. If we don’t have an index in a book and ask us to locate a specific chapter in that book, we will have to look at every page starting from the first page of the book.
3. On, the other hand, if we have the index, we look up the page number of the chapter in the index, and then directly go to that page number to locate the chapter.
4. Obviously, the book index is helping to drastically reduce the time it takes to find the chapter.
5. In a similar way, Table and View indexes can help the query to find data quickly.
6. In fact, the existence of the right indexes can drastically improve the performance of the query. If there is no index to help the query, then the query engine checks every row in the table from the beginning to the end. This is called a Table Scan. A table scan is bad for performance.

**What are the types of Indexes available in SQL Server?**

1. [**Clustered Index**](https://dotnettutorials.net/lesson/sql-server-clustered-index/)
2. [**Non-Clustered Index**](https://dotnettutorials.net/lesson/sql-server-non-clustered-index/)

[**What is a Clustered Index in SQL Server?**](https://dotnettutorials.net/lesson/sql-server-clustered-index/)

1. In the case of a [**clustered index**](https://dotnettutorials.net/lesson/sql-server-clustered-index/), the arrangement of the data in the index table will be the same as the arrangement of the data of the actual table.
2. Example: The index we find the start of a book.
3. When a table has a clustered index then the table is called a clustered table.
4. If a table has no clustered index its data rows are stored in an unordered structure.’
5. A table can have only one clustered index in it which will be created when the primary key constraint used in a table.
6. A clustered index determines the physical order of data in a table. For this reason, a table can have only one clustered index.

[**What is a Non-Clustered Index in SQL Server?**](https://dotnettutorials.net/lesson/sql-server-non-clustered-index/)

1. In the case of a [**non-clustered index**](https://dotnettutorials.net/lesson/sql-server-non-clustered-index/), the arrangement of data in the index table will be different from the arrangement of the actual table.
2. A non-clustered index is analogous to an index in a textbook. The data is stored in one place, the index is another place. The index will have pointers to the storage location of the data.
3. Since, the non-clustered index is stored separately from the actual data a table can have more than one non clustered index, just like how a book can have an index by chapters at the beginning and another index by common terms at the end.
4. In the index itself, the data is stored in an ascending or descending order of the index key which does not in any way influence the storage of data in the table.
5. In a table, we can create a maximum of 249 non clustered indexes.

**What is the difference between clustered and non-clustered index in SQL Server?**

This is one of the frequently asked SQL Server Indexes Interview Questions. Let us understand the differences.

1. Only one clustered index per table whereas we can have more than one non-clustered index.
2. **Clustered Index** is slightly faster than the Non-Clustered Index. This is because when a Non Clustered Index is used there is an extra look up from the Non-Clustered Index to the table, to fetch the actual rows.
3. The clustered index determines the storage order of rows in the table and hence does not require additional disk space whereas a non-clustered index is stored separately from the table, additional storage space is required.
4. **A clustered index** is a special type of index that reorders the way records in the table are physically stored. Therefore a table can have only one clustered index.
5. **A non-clustered index** is a special type of index in which the logical order of the index does not match the physical stored order of the rows on disk.

**What is a Unique Index in SQL Server?**

1. If the index is created by using the “UNIQUE” option then that column on which the index is created will not allow duplicate values i.e. it works as a unique constraint.
2. The unique constraint can be either unique clustered or unique non-clustered also.
3. While creating an index if clustered or non-clustered is not specified by default it is non-clustered.
4. A unique index is used to enforce uniqueness of key values in the index.

**What are the differences between UNIQUE Constraints and the UNIQUE Index?**

There are no major differences between a unique constraint and a unique index. In fact, when we add a unique constraint, a unique index gets created behind the scenes.

**ALTER TABLE tblEmployees ADD CONSTRAINT UQ\_tblEmplyees\_City UNIQUE (city)**

It will create a UNIQUE non clustered index on the City column. If we want we can also create a clustered unique index on City column like below

**ALTER TABLE tblEmployees ADD CONSTRAINT UQ\_tblEmplyees\_City UNIQUE CLUSTERED (city)**

It will create a clustered index on the City column.

**When should we be creating a unique constraint over a unique index?**

To make our intentions clear, create a unique constraint when data integrity is the objective. This makes the objective of the index very clear. In either case, data is validated in the same manner, and the query optimizer does not differentiate between a unique index created by a unique constraint or manually created.

**When SQL Server uses Indexes?**

1. SQL Server uses indexes of a table provided the select or update or delete statement contained “WHERE” condition in them and moreover the where condition column must be an indexed column.
2. If the select statement contains an “ORDER BY” clause also indexes will use.

NOTE: When SQL Server is searching for information under the database first it verifies the best execution plan for retrieving the data and uses that plan which can be either a full-page scan and index scan also.

**When should we create indexes on a table?**

1. We need to create an index on a table column provided those columns are frequently used in where condition or order by clause.
2. It is not advised creating an index on each and every column because a number of indexes can degrade the performance of the database also because every modification we make on the data should be reflected in all the index tables.

**What is a Covering query?**

1. If all the columns that we have requested in the select clause of the query are present in the index, then there is no need to look up in the table again. The requested column data can simply be returned from the index.
2. A clustered index always covers a query, since it contains all of the data in a table. A composite index is an index on two or more columns. Both clustered and non-clustered indexes can be composite indexes. To a certain extent, a composite index can cover a query.

**What is a table scan? Or what is the impact of table scans on performance?**

This is one of the frequently asked SQL Server Indexes Interview Questions. When SQL Server has no index to use for searching, the result is similar to the reader who looks at every page in a book to find a word. The SQL engine needs to visit every row in a table. In database terminology, we call this behavior a table scan or just scan. A full table scan of a very large table can degrade the performance. Creating proper indexes will allow the database to quickly narrow in on the rows to satisfy the query and avoid scanning every row in the table.

**What is the system stored procedure that can be used to list all the indexes that are created for a specific table?**

**sp\_helpindex** is the system stored procedure that can be used to list all the indexes that are created for a specific table. For example, to list all the indexes on table tblCustomers, we can use the following command.

**EXEC sp\_helpindex tblCustomers**

**What is the purpose of the query optimizer in SQL Server?**

An important feature of SQL Server is the query optimizer (component). The query optimizer’s job is to find the fastest and least resource-intensive means of executing incoming queries. An important part of this job is selecting the best index or indexes to perform the task.

**What is the first thing you will check for if the query below is performing very slowly?**

**SELECT \* FROM tblProducts ORDER BY UnitPrice ASC**

Check if there is an Index created on the UntiPrice column used in the ORDER BY clause. An index on the UnitPrice column can help the above query to find data very quickly. When we ask for sorted data, the database will try to find an index and avoid sorting the results during the execution of the query. We control sorting of data by specifying a field, or fields, in an ORDER BY clause, with the sort order as ASC (ascending) or DESC (descending).

With no index, the database will scan the **tblProducts** table and sort the rows to process the query. However, if there is an index, it can provide the database with a presorted list of prices. The database can simply scan the index from the first entry to the last entry and retrieve the rows in sorted order.

The same index works equally well with the following query, simply by scanning the index in reverse.

**SELECT \* FROM tblProducts ORDER BY UnitPrice DESC**

**What is the significance of an Index on the column used in the GROUP BY clause?**

Creating an Index on the column, that is used in the **GROUP BY** clause, can greatly improve the performance. We use a **GROUP BY** clause to group records and aggregate values, for example counting the number of products with the same UnitPrice. To process a query with a GROUP BY clause the database will often sort the results on the columns included in the GROUP BY.

The following query counts the number of products at each price by grouping together records with the same UnitPrice value.

**SELECT UnitPrice, Count(\*) FROM tblProducts GROUP BY UnitPrice**

The database can use the index (Index on **UNITPRICE** column) to retrieve the prices in order. Since matching prices appear in consecutive index entries, the database is able to count the number of products at each price quickly. Indexing a field used in a **GROUP BY** clause can often speed up a query.

**What is the role of an Index in maintaining a unique column in a table?**

1. Columns requiring unique values (such as primary key columns) must have a unique index applied. There are several methods available to create a unique index.
2. Marking a column as a primary key will automatically create a unique index on the column.
3. We can also create a unique index by checking the Create UNIQUE checkbox when creating the index graphically.
4. We can also create a unique index using SQL with the following command:

**CREATE UNIQUE INDEX IDX\_ProductName On Products (ProductName)**

The above SQL command will not allow any duplicate values in the ProductName column and an index is the best tool for the database to use to enforce this rule. Each time an application adds or modifies a row in the table, the database needs to search all existing records to ensure none of the values in the new data duplicate existing values.

**What are the disadvantages of an Index in SQL Server?**

This is one of the frequently asked SQL Server Indexes Interview Questions. There are 2 disadvantages of an Index

1. Increased Disk Space
2. Insert, Update and Delete statements could be slow. In short, all DML statements could be slow.

**Disk Space:** Indexes are stored on the disk and the amount of space required will depend on the size of the table and the number and types of columns used in the index. Disk space is generally cheap enough to trade for application performance, particularly when a database serves a large number of users.

**Insert, Update and Delete statements could be slow:** Another downside to using an index is the performance implication on data modification statements. Any time a query modifies the data in a table (**INSERT, UPDATE, or DELETE**), the database needs to update all of the indexes where data has changed. Indexing can help the database during data modification statements by allowing the database to quickly locate the records to modify, however, providing too many indexes to update can actually hurt the performance of data modifications. This leads to a delicate balancing act when tuning the database for performance.

**How many Clustered and Non-Clustered Indexes can you have per table?**

**Clustered Index**: Only one Clustered Index per table. A clustered index contains all of the data for a table in the index, sorted by the index key. The Phone Book is an example of the Clustered Index.

ADVERTISEMENT

**Non-Clustered Index**: We can have multiple Non-Clustered Indexes per table. Index at the back of a book is an example of a Non-Clustered Index.

For SQL Server 2005: 1 Clustered Index + 249 Nonclustered Index = 250 Index  
For SQL Server 2008: 1 Clustered Index + 999 Nonclustered Index = 1000 Index

**Which Index is faster, Clustered or Non Clustered Index?**

**Clustered Index** is slightly faster than the Non-Clustered Index. This is because when a Non Clustered Index is used there is an extra look up from the Non-Clustered Index to the table, to fetch the actual rows.

**When is it usually better to create a unique non-clustered index on the primary key column?**

Sometimes it is better to use a unique non-clustered index on the primary key column, and place the clustered index on a column used by more queries. For example, if the majority of searches are for the price of a product instead of the primary key of a product, the clustered index could be more effective if used on the price field.

**What is a Composite Index in SQL Server? or What is the advantage of using a Composite Index in SQL Server? or What is Covering Query?**

**A composite index** is an index on two or more columns. Both clustered and non-clustered indexes can be composite indexes.

If all of the information for a query can be retrieved from an Index then it is called as Covering query. A clustered index, if selected for use by the query optimizer, always covers a query, since it contains all of the data in a table.

**What are the different index configurations a table can have?**

A table can have one of the following index configurations:

1. **No indexes**
2. **A clustered index**
3. **A clustered index and many non-clustered indexes**
4. **A non-clustered index**
5. **Many non-clustered indexes**

**What is a table called, if it does not have neither Cluster nor Non-cluster Index? What is it used for?**

Unindexed table or **Heap**. Microsoft Press Books and Book On Line (BOL) refers to it as Heap. A heap is a table that does not have a clustered index and, therefore, the pages are not linked by pointers. The IAM pages are the only structures that link the pages in a table together.

Unindexed tables are good for fast storing of data. Many times it is better to drop all indexes from table and than do bulk of inserts and to restore those indexes after that.

**How to know which index a table is using?**

SELECT table\_name,index\_name FROM user\_constraints

# **SQL Server Triggers Interview Questions and Answers**

**What is a Trigger in SQL Server?**

A [**Trigger**](https://dotnettutorials.net/lesson/dml-triggers-in-sql-server/)is a database object which can also be treated as a special kind of stored procedure that automatically executes when language events (INSERT, DELETE or UPDATE) occurs. Triggers are stored in and managed by the DBMS. Triggers are similar to stored procedures in that both consist of procedural logic that is stored at the database level.

However, the difference between a trigger and a stored procedure is that the trigger is attached to a table and is only fired when an INSERT, UPDATE or DELETE operation occurred whereas stored procedures are not attached to a specific table and can be executed explicitly by making a call to the stored procedure. In addition to that, triggers can also execute stored procedures.

**Nested Trigger**: A trigger can also contain INSERT, UPDATE and DELETE logic within itself; so, when the trigger is fired because of data modification, it can also cause another data modification, thereby firing another trigger. A trigger that contains data modification logic within itself which causes another trigger to be fired is called the nested trigger.

**What are the types of Triggers available in SQL Server?**

The following are the two types of triggers in SQL Server.

1. **After Triggers (For Triggers):** Fired after Insert, Update and Delete operations on a table.
2. **Instead of Triggers:** Fired instead of Insert, Update and Delete operations on a table.

**In SQL Server, there are 4 types of triggers**

1. **DML Triggers – Data Manipulation Language**
2. **DDL Triggers – Data Definition Language**
3. **CLR triggers – Common Language Runtime**
4. **Logon triggers**

**What are the special tables used by Triggers in SQL Server?**

**Triggers** make use of two special tables called **inserted** and **deleted**. The **inserted** table contains the data referenced in an INSERT before it is committed to the database. The **deleted** table contains the data in the underlying table referenced in a DELETE before it is removed from the database. When an UPDATE is issued both tables are used. More specifically, the new data referenced in the UPDATE statement is contained in **the inserted**table and the data that is being updated is contained in **the deleted** table.

**What is MAGIC TABLE in triggers?**

These are the special kind of table which is created inside of a trigger when we perform insert, update and delete operations. The Magic tables are invisible tables or virtual tables. We can see them only with the help of TRIGGERS in SQL Server. The Magic tables are those tables that allow us to hold INSERTED, DELETED and UPDATED values during insert delete and update DML operations on a table in SQL Server.

Basically there are two types of magic tables in SQL Server namely INSERTED and DELETED magic table. The UPDATE can be performed with the help of these two.

**What is the INSERTED Magic Table?**

This table is created when we perform an insert operation that provides access to the values being inserted into the table.

Whenever we insert the values into a table those values we can see in the inserted magic table like below

CREATE TRIGGER T1 ON EMPLOYEE

FOR INSERT

AS

BEGIN

SELECT \* FROM inserted

END

**Let’s insert one record: INSERT INTO EMPLOYEE VALUES(106,’FF’, 30000, ‘UP’)**

**What is Deleted Magic Table in SQL Server?**

This table is created when we perform a delete operation providing access to the record being deleted. Whenever we delete a record from a table the record information can view with the deleted magic table like below.

CREATE TRIGGER T2 ON EMPLOYEE

FOR DELETE

AS

BEGIN

SELECT \* FROM deleted

END

**Let’s delete one record: DELETE FROM EMPLOYEE WHERE EID = 105**

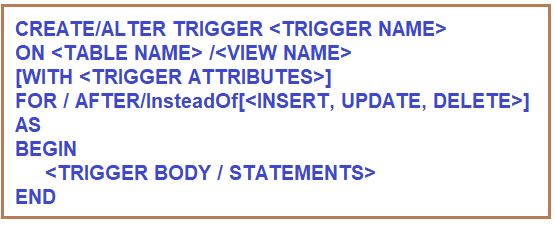
**What is a DML Trigger in SQL Server?**

DML stands for Data Manipulation Language. INSERT, UPDATE, and DELETE statements are DML statements. The DML triggers are fired whenever data is modified using INSERT, UPDATE, and DELETE events.

These DML Triggers execute when the user tries to modify or change data through data manipulation language events such as INSERT, UPDATE and DELETE statements on the table or view.

DML Triggers can be used to enforce business rules and data integrity. DML triggers are similar to constraints in the way they enforce integrity.

**Syntax:**



**DML triggers can be again classified into 2 types.**

1. After trigger (Sometimes called as FOR triggers)
2. Instead of trigger

**After triggers as the name says fires after the triggering action**. The INSERT, UPDATE, and DELETE statements cause an after trigger to fire after the respective statements complete execution.

**On the other hand, as the name says, INSTEAD of triggers, fires instead of the triggering action**. The INSERT, UPDATE, and DELETE statements, can cause an INSTEAD OF trigger to fire INSTEAD OF the respective statement execution.

**How to view the updating data in a table?**

When we perform an update operation we will be having both inserted and deleted tables where inserted will provide access to the new values being inserted and deleted table provides access to the old values of the table.

Whenever we update a record data in the table, we can view the new value in the inserted magic table and old value in the deleted magic table like below.

CREATE TRIGGER T3 ON EMPLOYEE

FOR UPDATE

AS

BEGIN

SELECT \* FROM deleted

SELECt \* FROM inserted

END

**Let’s delete one record**

**UPDATE EMPLOYEE SET ENAME =’PRANAYA’ WHERE EID = 101**

**What are DDL triggers in SQL Server?**

DDL triggers fire in response to DDL events – CREATE, ALTER, and DROP (Table, Function, Index, Stored Procedure, etc…).

The DDL triggers fires in response to a variety of data definition language events such as Create, Alter, Drop, Grant, Denay and Revoke. The DDL triggers are introduced from SQL Server 2005 version which will be used to restrict DDL operations such as CREATE, ALTER and DROP commands.

A DDL trigger is a special type of stored procedure that executes in response to a server scoped or database scoped events. DDL triggers fire only after the DDL statements execute so we cannot use “InsteadOf Triggers” here and moreover DDL triggers will not fire in response to events that affect local temporary tables.

**Syntax:**

CREATE TRIGGER **<**TRIGGER NAME**>**

ON ALL SERVER/ DATABASE

**[**with trigger attributes**]**

FOR / ALTER **<**Event\_Type**>**

AS

BEGIN

**<**TRIGGER BODY/ STATEMENTS**>**

END

**<Event\_Type>** refers to the event that will fire the trigger which can be anything like Create\_Table, Drop\_Table, Alter\_Table, etc.

**What is the use of DDL triggers in SQL Server?**

If we want to execute some code in response to a specific DDL event. To prevent certain changes to our database schema. Audit the changes that the users are making to the database structure

**How to Enable, Disable and Drop Triggers in SQL Server?**

**To disable the trigger:**

1. Right-click on the trigger in object explorer and select **“Disable”** from the context menu
2. We can also disable the trigger using the following T-SQL command: **DISABLE TRIGGER trMyFirstTrigger ON DATABASE**

**To enable the trigger:**

1. Right-click on the trigger in object explorer and select “**Enable**” from the context menu
2. We can also enable the trigger using the following T-SQL command: **ENABLE TRIGGER trMyFirstTrigger ON DATABASE**

**To drop trigger:**

1. Right-click on the trigger in object explorer and select “**Delete**” from the context menu
2. We can also drop the trigger using the following T-SQL command: **DROP TRIGGER trMyFirstTrigger ON DATABASE**

# **SQL Server Views Interview Questions**

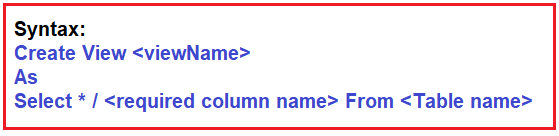
**What is a View in SQL Server?**

A view is nothing more than a saved SQL query. A view can also be considered as a virtual table. So, we can think of a view either as a compiled SQL query or a virtual table. As a view represents a virtual table it does not physically store any data by default. When we query a view we actually, retrieve the data from the underlying database tables.

**What are the differences between a table and a view?**

When compared with a table we have the following differences between a table and view.

1. The table is physical and the view is logical
2. A table is an independent object whereas view is a dependent object that is a view depends on a table or tables from which it is loading the data.
3. When a new table is created from an existing table the new and old tables are independent themselves that is the changes of one table will not be reflected into the other table whereas if a view is created based on a table any changes that are performed on the table reflects into the view and any changes performed on the view reflected in the table also.



**How many types of views are there in SQL Server?**

We can create the view in two ways those are

1. Simple view and Updatable views
2. Complex view and non-updatable views.

**What is a simple view or Updatable view?**

1. The view which is created basing on the columns of a single table is known as the simple view.
2. We can perform all DML operations on a simple view so that a simple view is also called an updatable view or dynamic view.

**What is a complex View in SQL Server?**

1. When we create a view on more than 1 table then it is known as the complex view.
2. On a complex view, we cannot perform DML operations so that a complex view is also called a non-updatable or static view.

**Can we drop a table that has dependent views on it?**

Yes, we can drop a table even if any dependent views are associated with it, but the views that are associated with it will not be dropped. They will still execute in the database only with the status as inactive object and all those views become active and start functioning provided the table is recreated.

**Can we create a view based on other views?**

Yes, we can create a view based on other views. Usually, we create views based on tables, but it is also possible to create views based on views.

**Can we update the views?**

Yes, views can be updated. However, updating a view that is based on multiple tables, may not update the underlying tables correctly. To correctly update a view that is based on multiple tables we can make use “Instead OF triggers” in SQL Server.

**Why do we need Views in SQL Server?**

To protect the data. If we have a table containing sensitive data in certain columns, we might wish to hide those columns from certain groups of users. For instance, customer names, addresses, and social security numbers might all be stored in the same table; however, for lower-level employees like shipping clerks, you can create a view that only displays customer name and address. You can grant permissions to a view without allowing users to query the original tables.

A view is a logical table but what it stores internally is a select statement that is used for creating the view. So that whenever a user performs an operation on the view like select, insert, update or delete internally the view performs those operations on a table.

Simply we can say that view will act as an interface between the data provider (Table) and the User.

A view is created based on a table any changes that are performed on the table reflect into the view any changes performed on the view reflected in the table also.

**What are the advantages of using views? OR when do you usually use views?**

**Advantages of using views:**

Views can be used to reduce the **complexity of the database schema**, for non-IT users. The sample view, **vWEmployeesByDepartment**, hides the complexity of joins. Non-IT users find it easy to query the view, rather than writing complex joins.

Views can be used as a mechanism to implement **row and column level security**.

**Row Level Security:**

For example, I want an end-user, to have access only to IT Department employees. If I grant him access to the underlying tblEmployees and tblDepartments tables, he will be able to see, every department employee. To achieve this, I can create a view, which returns only IT Department employees, and grants the user access to the view and not to the underlying table.

**Column Level Security:**

Salary is confidential information and I want to prevent access to that column. To achieve this, we can create a view, which excludes the Salary column, and then grant the end-user access to these views rather than the base tables.

Views can be used to present **only aggregated data** and **hide detailed data**. **The view that returns summarized data**, Total number of employees by Department.

**What are indexed views? Or What are materialized views?**

A view is a virtual table that means it does not contain any physical data. A view is nothing more than a compiled SQL query. Every time, we issue a select query against a view, we actually get the data from the underlying base tables and not from the view, as the view itself does not contain any data.

When we create an index on a view, the data gets physically stored in the view. So, when we issue a select query against an indexed view, the data is retrieved from the index without having to go to the underlying table, which will make the select statement to work slightly faster. However, the disadvantage is INSERT, UPDATE and DELETE operations will become a little slow, because every time we insert or delete a row from the underlying table, the view index needs to be updated. In short, DML operations will have a negative impact on performance.

Oracle refers to indexed views as materialized views.

Only the views created with schema binding can have an Index. Simply adding WITH SCHEMABINDING to the end of the CREATE VIEW statement will accomplish this. However, the effect is that any changes to the underlying tables which will impact the view are not allowed. Since the indexed view is stored physically, any schema changes would impact the schema of the stored results set. Therefore, SQL Server requires that schema binding is used to prevent the view’s schema (and therefore the underlying tables) from changing.

The first index for a view must be a UNIQUE CLUSTERED INDEX, after which, it’s possible to create non-clustered indexes against the view.

Indexed Views are heavily used in data warehouses and reporting databases that are not highly transactional.

**What are the limitations of a View?**

1. We cannot pass parameters to a view.
2. Rules and Defaults cannot be associated with views.
3. The ORDER BY clause is invalid in views unless TOP or FOR XML is also specified.
4. Views cannot be based on temporary tables.

# **SQL Server Functions Interview Questions and Answers**

**What is a function in SQL Server?**

A function is a database object in SQL Server. Basically, it is a set of SQL statements that accept only input parameters, perform actions, and return the result. The function can return only a single value or a table. We can’t use the function to Insert, Update, and Delete records in the database table(s).

1. It is also a subprogram like a stored procedure that is defined for performing an action such as complex calculation and returns the result of the action as a value.
2. These functions are created by the user or programmer.
3. Functions are taking some parameters, do some processing, and returning some results back

**For example Select SQUARE(3) Output: 9**

Some functions also do not take any parameters. **For Example: Select GETDATE()**

So, we can say that a function can have the parameter that is optional but a function should return a value that is mandatory.

**Types of User-Defined Functions in SQL Server:**

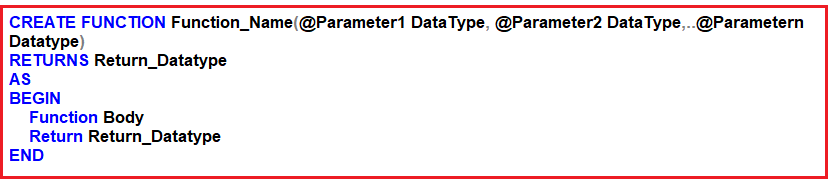
In SQL Server, there are 3 types of User-Defined Functions

1. Scalar functions
2. Inline table-valued functions
3. Multi-statement table-valued functions

**What is a scalar function in SQL Server?**

The functions which return a single value are known as scalar value function. The **Scalar functions** may or may not have parameters, but always return a single (scalar) value. The returned value can be of any data type, except **text, ntext, image, cursor, and timestamp**.

**To create a function, we use the following syntax:**

****

**What is a table-valued function?**

In this case, we can return a table as an output from the function. These are again of two types

1. Inline Table-valued Function
2. Multi-statement table value function

**Where can we use the Inline Table-Valued function?**

The Inline Table-Valued functions can be used to achieve the functionality of parameterized views. The table returned by the table-valued function can also be used in joins with other tables.

**What is Inline table-valued functions?**

In this case, the body of the function will have only a single select statement prepared with the “RETURN” statement. The syntax for creating a table value function

CREATE/ALTER FUNCTION **<**FUNCTION NAME**>(**@**<**VARIABLE NAME**><**DATA TYPE**>** **[**SIZE**])**

RETURNS TABLE

AS

RETURN **<**SELECT STATEMENT**>**

1. We specify the Table as the return type instead of any scalar data type.
2. The function body is not closed between BEGIN and END block
3. The structure of the table that gets returned is determined by the select statement within the function.

**What are Multi-Statement Table-Valued Functions in SQL Server?**

This is the same as the inline table-valued function which can return a table as an output but here the body can contain more than one statement and also the structure of the table being returned can be defined. The Syntax is given below.

CREATE/ALTER FUNCTION **<**FUNCTIONNAME**>** **(**@**<**PARAM**><**DATATYPE**>** **[**SIZE**]**.....**)**

RETURNS @**<**TABLE VAR**>** TABLE **(<**COLUMN DEFINITIONS**>)**

**[**WITH **<**FUNCTION ATTRIBUTES**>]**

AS

BEGIN

**<**FUNCTION BODY**>**

RETURN

END

**Note:** In case of a multi-statement table-valued function we need to define our own structure to the table being return.

**Differences between Inline Table-Valued functions and Multi-statement Table-Valued functions**

In an Inline Table-Valued function, the RETURNS clause cannot contain the structure of the table, the function returns. Whereas, with the multi-statement table-valued function, we specify the structure of the table that gets returned

Inline Table-Valued function cannot have BEGIN and END block as it returns a single select statement, whereas the multi-statement function can have the begin and end block as it contains more than one select statement.

Inline Table-valued functions are better for performance than multi-statement table-valued functions. If the given task, can be achieved using an inline table-valued function, always prefer to use them, over multi-statement table-valued functions.

It’s possible to update the underlying table, using an inline table-valued function, but not possible using a multi-statement table-valued function.

**Reason for improved performance of an inline table-valued function:**

Internally, SQL Server treats an inline table-valued function much like it would a view and treats a multi-statement table-valued function similar to how it would a stored procedure.

**What are the differences between functions and procedures in SQL Server?**

Stored Procedures are pre-compiled objects which are compiled for the first time and its compiled format is saved which executes (compiled code) whenever it is called. But Function is compiled and executed every time when it is called.

**BASIC DIFFERENCE**

1. The function must return a value but in Stored Procedure, it is optional (Procedure can return zero or n values).
2. Functions can have only input parameters whereas Procedures can have input/output parameters.
3. A Function can be called from Procedure whereas Procedures cannot be called from Function
4. From a procedure, we can call another procedure or a function whereas from a function we can call another function but not a procedure.

**ADVANCE DIFFERENCE**

1. The procedure allows SELECT as well as DML (INSERT/UPDATE/DELETE) statement in it whereas Function allows only SELECT statement in it.
2. Procedures cannot be utilized in a SELECT statement whereas Function can be embedded in a SELECT statement.
3. Stored Procedures cannot be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section whereas Function can be.
4. Functions that return tables can be treated as a row set. This can be used in JOINs with other tables
5. The exception can be handled by a try-catch block in a procedure whereas a try-catch block cannot be used in a Function.
6. We can go for Transaction Management in Procedure whereas we can’t go into Function.
7. We call a procedure using EXECUTE/ EXEC command whereas a function is called by using the SELECT command only.
8. Stored procedures support deferred name resolution. For example, while writing a stored procedure that uses table names, for example, table1, table2, etc. but these tables do not exist in the database is allowed during the creation of the stored procedure but runtime throws error whereas functions do not support deferred name resolution.

# **SQL Server TCL Interview Questions**

**What is Transaction Management?**

The process of combining a set of related operations into a single unit and executing those operations by applying to do everything or do nothing principle is called transaction management. For example, transfer money task is the combination of two operations

1. Withdraw money from the source account
2. Deposit amount into the destination account

We need to execute these two statements by applying to do everything or do nothing principle performing transaction management. Every transaction has two boundaries

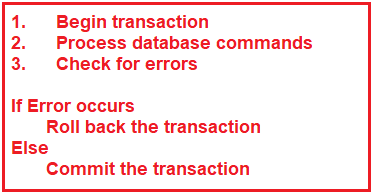
1. Beginning
2. Ending

Controlling the boundaries of a transaction is nothing but transaction management.

**What is a Transaction?**

A transaction is a unit of work or set of DML statements (i.e. INSERT, UPDATE and DELETE) which should be executed as one unit. A transaction ensures that either all of the command succeeds or none of them. If one of the commands in the transaction fails, all of the commands fail and any data that is modified in the database is rolled back. In this way, transaction maintains the integrity of data in a database.

Transaction processing work as follows:



The rules of transaction management tell that either all the statements in the transaction should be executed successfully or none of those statements to be executed. To manage transaction we have provided with transaction control language that provides commands like “**COMMIT TRANSACTION”, “ROLLBACK TRANSACTION”, and “SAVE TRANSACTION**”.

1. **COMMIT TRANSACTION:**It indicates that the transaction completed successfully and all the data manipulation operation performed since the start of the transaction committed to the database and frees the resources held by the transaction.
2. **ROLLBACK TRANSACTION:**It will bring back an implicit or explicit transaction to the beginning of the transaction or erase all data modifications made from resources held by the transaction.
3. **SAVE TRANSACTION:** This is used for dividing or breaking a transaction into multiple units so that we will have a chance of rollbacking a transaction up to a location.

**Explain various types of SQL Server transactions modes.**

SQL Server transactions are classified into three types, they are as follow**s**

1. Auto Commit Transaction Mode (default)
2. Implicit Transaction Mode
3. Explicit Transaction Mode

**Auto Commit Transaction Mode in SQL Server:**

1. In this mode of transaction, programmers are not responsible for beginning a transaction or ending a transaction. Whenever we perform or execute any DML statement SQL Server will only begin the transaction and end the transaction with a Commit or Rollback. So the programmer does not have any control over them.
2. This is the default mode of transaction. In this mode, if the transaction is completed successfully, it is committed. If the transaction faces any error, it is rolled back.
3. In auto-commit mode, each SQL statement is treated as a separate transaction.

**Implicit Transaction Mode in SQL Server:**

1. In this mode of transaction SQL Server begins a transaction implicitly before the execution of any DML statement and developers are responsible to end the transaction with a commit or rollback. Once a transaction ends, automatically a new transaction start.
2. When the transaction is in implicit mode, a new transaction starts automatically after the current transaction is committed or rolled back. Nothing needs to be done to define the start of the transaction. It generates a continues chain of transactions.
3. Implicit mode for transactions in SQL Server is set to ON or OFF by using the command SET IMPLICIT\_TRANSACTIONS to ON or OFF. If the implicit mode is set to ON when a new transaction is implicitly started whenever any particular SQL statement is executed. The particular SQL statement includes INSERT, SELECT, DELETE, AND UPDATE and other SQL statements as well. It is called implicit mode because of the fact that once IMPLICIT\_TRANSACTIONS is set to ON then the transactions are created implicitly for certain SQL statements without having to say we want to create a transaction each time.

**Explicit Transaction Mode in SQL Server:**

1. In this mode of transaction, a programmer is only responsible for beginning the transaction and ending the transaction.
2. Transactions that have a START and END explicitly written by the programmer are called as an explicit transaction.
3. In the explicit mode of transaction, every transaction starts with a BEGIN TRANSACTION statement and ends with either a ROLLBACK TRANSACTION statement (when the transaction does not complete successfully) or a COMMIT TRANSACTION statement (when the transaction completes successfully). The explicit mode is most commonly used in triggers, stored procedures and application programs.

**What is Transaction Control Language?**

A **Transaction Control Language** (**TCL**) is a computer language and a subset of SQL which is used to control transactional processing in a database. A transaction is a logical unit of work that comprises one or more SQL statements, usually, a group of Data Manipulation Language (DML) statements.

**Why we need Transaction Control Language?**

1. To safeguard enterprise data (to make enterprise data consistent and to achieve data integrity).
2. A transaction is the propagation of one or more changes to the database. For example, if we are creating a record or updating a record or deleting a record from the table, then we are performing a transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors.

**What is the rule of Transaction?**

The rule of transaction tells that either all the statements in the transaction should be executed successfully or none of those statements to be executed.

**What has distributed transactions in SQL Server? When are they used?**

A distributed transaction is one in which it updates data present in two or more databases in a SQL Server. The management of such transactions is done by a component called the transaction manager. They are useful in updating data that is distributed. Distributed transactions must be used when real-time updates are required simultaneously on multiple databases.

**What is a nested transaction?**

A nested transaction is one in which a new transaction is started by an instruction that is already inside another transaction. This new transaction is said to be nested. The isolation property of the transaction is obeyed here because the changes made by the nested transaction are not seen or interrupted by the host transaction.

**When to Use Transactions?**

We should use transactions when several operations must succeed or fail as a unit. The following are some frequent scenarios where the use of transactions is recommended:

1. In batch processing where multiple rows must be inserted, updated, or deleted as of a single unit
2. Whenever a change to one table requires that other tables be kept consistent
3. When modifying data in two or more databases concurrently
4. In distributed transactions where data is manipulated in databases on various servers
5. When we use transactions, we put locks on data that is pending for the permanent change to the database. No other operations can take place on locked data until the acquired lock is released. We could lock anything from a single row up to the entire database. This is called concurrency, which means how the database handles multiple updates at one time.

**Note:** it’s important to keep transactions pending for the shortest period of time. A lock stops others from accessing the locked database resource. Too many locks, or locks on frequently accessed resources, can seriously degrade performance.

**How to implement/ control transactions in SQL Server?**

The following commands are provided by TCL to control transactions:

1. **BEGIN TRANSACTION:**To start the transaction
2. **COMMIT:** to save the changes.
3. **ROLLBACK TRANSACTION:** to roll back the changes.
4. **SAVE TRANSACTION:** creates points within groups of transactions in which to ROLLBACK

Transnational control commands are only used with the DML commands INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed to the database.

**What is the need of BEGIN TRANSACTION in SQL Server?**

The BEGIN TRANSACTION is used to start the transaction and also used to add a nested transaction. The Syntax is given below.

**BEGIN TRANSACTION**  
**<WRITE STATEMENTS>**

**What is the need for the COMMIT command in SQL Server?**

1. COMMIT command is used to end the transaction and save the data permanently to the database or it is used to make the transaction permanent so we cannot undo or recall the records.
2. COMMIT is used for saving the data that has been changed permanently because whenever we perform any DML operations like INSERT, UPDATE or DELETE then we are required to write commit at the end of all or every DML operations in order to save it permanently.
3. If we are not writing COMMIT then our data will be restored to its previous position.
4. The COMMIT command saves all transactions to the database since the last COMMIT or ROLLBACK command.

**Syntax:**  
**BEGIN TRANSACTION**  
**COMMIT TRANSACTION**

**What is the need for the ROLLBACK command in SQL Server?**

1. The ROLLBACK command is used to undo the transactions that have not already been saved to the database and get back to the initial state where the transaction started.
2. Whereas if we want to restore our data into its previous condition then we can write ROLLBACK at any time after the DML queries have been written but remember once COMMIT has been written then we cannot ROLLBACK the data.
3. More ever we can only ROLLBACK the DML query that has been written after the last COMMIT statement.
4. The concept of ROLLBACK and COMMIT is designed for data consistency because many users manipulate the data of the same table using the same database so the user must get the updated data. That’s why COMMIT and ROLLBACK are used.

**Syntax:**  
**BEGIN TRANSACTION**  
**ROLLBACK TRANSACTION**

**What is the need to SAVE TRANSACTION?**

1. SAVE TRANSACTION is used for dividing (or) breaking a transaction into multiple units. So that the user has a chance of rollbacking a transaction up to a location.
2. When a user sets a SAVE TRANSACTION within a transaction the save transaction defines a location to which a transaction can return if part of the transaction conditionally canceled.
3. If a transaction is rolled back to a save point, it must proceed to completion of the transaction with a commit statement or rollback statement.

**What are the properties of a transaction?**

A transaction is a group of SQL statements that are treated as a single unit. A successful transaction must pass the “ACID” test i.e. it must be

**Atomicity:**

A transaction is an atomicity if it works like a single action rather than a collection of separate operations. So, when all the separate operations succeed then the transaction succeeds and is committed to the database. On the other hand, if a single operation fails during the transaction then everything is considered to have failed and must be undone (rolled back) if it has already taken place but in any case not left half-done. Example of transferring money from one account to another.

That means this property of a transaction ensures that a transaction either completely or does not happen at all.

**Consistency:**

This property ensures that the database data is in the consistent state before the transaction and left in a consistent state after the transaction. If the transaction violates the rules it must be rolled back.

**Isolation:**

Every transaction has a well-defined boundary. That is it is isolated from another transaction. One transaction shouldn’t affect other transactions running at the same time. This prevents transactions from making changes to data based on the uncommitted information. Data modifications made by one transaction must be isolated from the data modifications made by all other transactions. Most databases use locking to maintain transaction isolation. That means a transaction should be isolated means that no other operation should be allowed to access or see the intermediate state data.

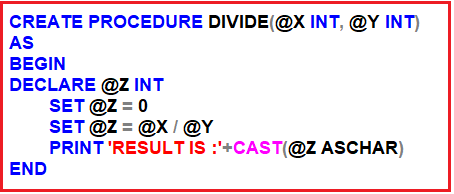
**Durability:**

Data modifications that occur within a successful transaction are kept permanently within the database in case of system failure.

## **SQL Server Exception Handling Interview Questions And Answers**

**Why we need Exception Handling in SQL Server?**

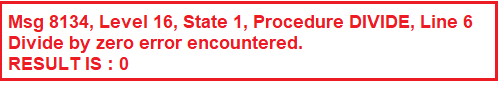
Let’s understand the need for Exception handling with an example. Let us create a stored procedure for dividing two numbers as shown below.



**Execution:**  
**EXEC DIVIDE 100, 5**  
**EXEC DIVIDE 100, 0**

In the above case when we execute the procedure we will get the error at the second execution because the divisor value is ZERO where a number cannot be divided by zero as per the rule of mathematics.

In SQL Server whenever an error occurs at a line, it will first print the error message and then continues with the execution so in the above case when the error occurred in the procedure the result will be as following



The problem in the above execution is even if the error occurred in the program, it still showing the result so there are chances of users being confused.

**Note:** Whenever an error occurred while executing a program developed by using any programming language the program terminates abnormally on the line where the error got occurred but in SQL Server it will still continue the program execution.

**What is Exception Handling?**

We handle errors of a program both in a programming language as well as databases also whereas handling errors in a programming language needs stopping the abnormal termination and allowing the statements which are not related to the error to execute but handling an error in SQL Server means stopping the execution of the statements which are related with the error.

**With the introduction of Try/Catch blocks in SQL Server 2005**, error handling in SQL Server is now similar to programming languages like C#, and java. Before understanding error handling using try/catch, let’s step back and understand how error handling was done in SQL Server 2000, using system function **@@Error**. Sometimes, system functions that begin with two at signs **(@@)**, are called **global variables**. They are not variables and do not have the same behaviors as variables, instead, they are very similar to functions.

Now let’s create **tblProduct** and **tblProductSales**that we will be using for the rest of this demo.

--SQL script to create tblProduct

Create Table tblProduct

**(**

ProductId **int** NOT **NULL** primary key,

Name nvarchar**(**50**)**,

UnitPrice **int**,

QtyAvailable **int**

**)**

Go

-- SQL script to load data **into** tblProduct

Insert **into** tblProduct values**(**1, 'Laptops', 2340, 100**)**

Insert **into** tblProduct values**(**2, 'Desktops', 3467, 50**)**

Go

-- SQL script to create tblProductSales

Create Table tblProductSales

**(**

ProductSalesId **int** primary key,

ProductId **int**,

QuantitySold **int**

**)**

Go

--Create a procedure

Create Procedure spSellProduct

@ProductId **int**,

@QuantityToSell **int**

As

Begin

-- Check the stock available, **for** the product we want to sell

Declare @StockAvailable **int**

Select @StockAvailable = QtyAvailable

**from** tblProduct **where** ProductId = @ProductId

-- Throw an error to the calling application, **if** enough stock **is** not available

If**(**@StockAvailable**<** @QuantityToSell**)**

Begin

Raiserror**(**'Not enough stock available',16,1**)**

End

-- If enough stock available

Else

Begin

Begin Transaction

-- First reduce the quantity available

Update tblProduct **set** QtyAvailable = **(**QtyAvailable - @QuantityToSell**)**

**where** ProductId = @ProductId

Declare @MaxProductSalesId **int**

-- Calculate MAX ProductSalesId

Select @MaxProductSalesId = Case When

MAX**(**ProductSalesId**)** IS **NULL**

Then 0 **else** MAX**(**ProductSalesId**)** end

**from** tblProductSales

-- Increment @MaxProductSalesId **by** 1, so we don't **get** a primary key violation

Set @MaxProductSalesId = @MaxProductSalesId + 1

Insert **into** tblProductSales values**(**@MaxProductSalesId, @ProductId, @QuantityToSell**)**

Commit Tran

End

End

**Stored procedure – spSellProduct**, has 2 parameters – **@ProductId** and **@QuantityToSell**. @ProductId specifies the product that we want to sell, and @QuantityToSell specifies the quantity we would like to sell.

In the procedure, we are using **Raiserror**() function to return an error message back to the calling application, if the stock available is less than the quantity we are trying to sell. We have to pass at least 3 parameters to the Raiserror() function.

**RAISERROR(‘Error Message’, ErrorSeverity, ErrorState)**

**Severity and State** are integers. In most cases, when you are returning custom errors, the severity level is 16, which indicates general errors that can be corrected by the user. In this case, the error can be corrected, by adjusting the **@QuantityToSell**, to be less than or equal to the stock available. ErrorState is also an integer between 1 and 255. RAISERROR only generates errors with the state from 1 through 127.

The problem with this procedure is that the **transaction is always committed**. Even, if there is an error somewhere, between updating **tblProduct** and **tblProductSales** table. In fact, the main purpose of wrapping these 2 statements (Update tblProduct Statement & Insert into tblProductSales statement) in a transaction is to ensure that, both of the statements are treated as a single unit. For example, if we have an error when executing the second statement, then the first statement should also be rolled back.

In SQL server 2000, to detect errors, we can use **@@Error** system function. @@Error returns a NON-ZERO value, if there is an error, otherwise ZERO, indicating that the previous SQL statement encountered no errors. The stored procedure **spSellProductCorrected**, makes use of @@ERROR system function to detect any errors that may have occurred. If there are errors, roll back the transaction, else commit the transaction. If you comment the line (Set @MaxProductSalesId = @MaxProductSalesId + 1) and then execute the stored procedure there will be a primary key violation error, when trying to insert into **tblProductSales**. As a result of this, the entire transaction will be rolled back.

Alter Procedure spSellProductCorrected

@ProductId **int**,

@QuantityToSell **int**

As

Begin

-- Check the stock available, **for** the product we want to sell

Declare @StockAvailable **int**

Select @StockAvailable = QtyAvailable

From tblProduct **where** ProductId = @ProductId

-- Throw an error to the calling application, **if** enough stock **is** not available

If**(**@StockAvailable**<** @QuantityToSell**)**

Begin

Raiserror**(**'Not enough stock available',16,1**)**

End

-- If enough stock available

Else

Begin

Begin Tran

-- First reduce the quantity available

Update tblProduct **set** QtyAvailable = **(**QtyAvailable - @QuantityToSell**)**

**where** ProductId = @ProductId

Declare @MaxProductSalesId **int**

-- Calculate MAX ProductSalesId

Select @MaxProductSalesId = Case When

MAX**(**ProductSalesId**)** IS **NULL**

Then 0 **else** MAX**(**ProductSalesId**)** end

From tblProductSales

-- Increment @MaxProductSalesId **by** 1, so we don't **get** a primary key violation

Set @MaxProductSalesId = @MaxProductSalesId + 1

Insert **into** tblProductSales values**(**@MaxProductSalesId, @ProductId, @QuantityToSell**)**

**if(**@@ERROR **<>** 0**)**

Begin

Rollback Tran

Print 'Rolled Back Transaction'

End

Else

Begin

Commit Tran

Print 'Committed Transaction'

End

End

End

**Note:**

@@ERROR is cleared and reset on each statement execution. Check it immediately following the statement being verified, or save it to a local variable that can be checked later.

In **the tblProduct** table, we already have a record with **ProductId = 2**. So the insert statement causes a primary key violation error. @@ERROR retains the error number, as we are checking for it immediately after the statement that causes the error.

Insert **into** tblProduct values**(**2, 'Mobile Phone', 1500, 100**)**

**if(**@@ERROR **<>** 0**)**

Print 'Error Occurred'

Else

Print 'No Errors'

On the other hand, when you execute the code below, you get a message **‘No Errors’**printed. This is because the @@ERROR is cleared and reset on each statement execution.

Insert **into** tblProduct values**(**2, 'Mobile Phone', 1500, 100**)**

--At this point @@ERROR will have a NON ZERO **value**

Select \* **from** tblProduct

--At this point, @@ERROR gets reset to ZERO, because of the

--**select** statement successfully executed

**if(**@@ERROR **<>** 0**)**

Print 'Error Occurred'

Else

Print 'No Errors'

In this example, we are storing the value of the @@Error function to a local variable, which is then used later.

Declare @Error **int**

Insert **into** tblProduct values**(**2, 'Mobile Phone', 1500, 100**)**

Set @Error = @@ERROR

Select \* **from** tblProduct

**if(**@Error **<>** 0**)**

Print 'Error Occurred'

Else

Print 'No Errors'

**Error handling in SQL Server 2005 and later versions:**

**Syntax:**

BEGIN TRY

**{** Any **set** of SQL statements **}**

END TRY

BEGIN CATCH

**[** Optional: Any **set** of SQL statements **]**

END CATCH

**[**Optional: Any other SQL Statements**]**

**Any set of SQL statements**, that can possibly throw an exception are wrapped between BEGIN TRY and END TRY blocks. If there is an exception in the TRY block, the control immediately jumps to the CATCH block. If there is no exception CATCH block will be skipped and the statements after the CATCH block are executed.

**Errors trapped by a CATCH block are not returned to the calling application**. If any part of the error information must be returned to the application the code in the CATCH block must do so by using RAISERROR() function.

**In procedure spSellProduct**, Begin Transaction and Commit Transaction statements are wrapped between Begin Try and End Try block. If there are no errors in the code that is enclosed in the TRY block, then COMMIT TRANSACTION gets executed and the changes are made permanent. On the other hand, if there is an error then the control immediately jumps to the CATCH block. In the CATCH block, we are rolling the transaction back. So it’s much easier to handle errors with Try/Catch construct than with @@Error system function.

Also notice that in the scope of the CATCH block, there are several system functions that are used to retrieve more information about the error that occurred these functions return NULL if they are executed outside the scope of the CATCH block.

**TRY/CATCH cannot be used in a user-defined function.**

Create Procedure spSellProduct

@ProductId **int**,

@QuantityToSell **int**

As

Begin

-- Check the stock available, **for** the product we want to sell

Declare @StockAvailable **int**

Select @StockAvailable = QtyAvailable

From tblProduct **where** ProductId = @ProductId

-- Throw an error to the calling application, **if** enough stock **is** not available

If**(**@StockAvailable**<** @QuantityToSell**)**

Begin

Raiserror**(**'Not enough stock available',16,1**)**

End

-- If enough stock available

Else

Begin

Begin Try

Begin Transaction

-- First reduce the quantity available

Update tblProduct **set** QtyAvailable = **(**QtyAvailable - @QuantityToSell**)**

**where** ProductId = @ProductId

Declare @MaxProductSalesId **int**

-- Calculate MAX ProductSalesId

Select @MaxProductSalesId = Case When

MAX**(**ProductSalesId**)** IS **NULL**

Then 0 **else** MAX**(**ProductSalesId**)** end

From tblProductSales

--Increment @MaxProductSalesId **by** 1, so we don't **get** a primary key violation

Set @MaxProductSalesId = @MaxProductSalesId + 1

Insert **into** tblProductSales values**(**@MaxProductSalesId, @ProductId, @QuantityToSell**)**

Commit Transaction

End Try

Begin Catch

Rollback Transaction

Select

ERROR\_NUMBER**()** **as** ErrorNumber,

ERROR\_MESSAGE**()** **as** ErrorMessage,

ERROR\_PROCEDURE**()** **as** ErrorProcedure,

ERROR\_STATE**()** **as** ErrorState,

ERROR\_SEVERITY**()** **as** ErrorSeverity,

ERROR\_LINE**()** **as** ErrorLine

End Catch

End

End

**How to handle errors in SQL Server?**

From SQL Server 2005 we are provided with a structure error handling mechanism with the help of TRY and CATCH blocks which should be used as follows

**Syntax:**

BEGIN TRY

--statements which will cause the error

--statements which are related to the error that should not execute **when** the error occurred

END TRY

BEGIN CATCH

--statements which should be executed **when** the error occurs

END CATCH

To overcome the problem we faced in the dividing procedure rewrite the procedure as follows.

**Example 1: Create a procedure for dividing 2 variables values by using TRY CATCH implementation with user-defined error statements.**

CREATE PROCEDURE SPDIVIDE**(**@X INT, @Y INT**)**

AS

BEGIN

DECLARE @Z INT

SET @Z = 0

BEGIN TRY

SET @Z = @X / @Y

PRINT 'RESULT IS :- '+ CAST**(**@Z AS CHAR**)**

END TRY

BEGIN CATCH

PRINT 'SECOND NUMBER SHOULD NOT BE ZERO'

END CATCH

END

**EXEC SPDIVIDE 10, 2**  
**OUTPUT: RESULT IS:- 5**  
**EXEC SPDIVIDE 10, 0**  
**OUTPUT: SECOND NUMBER SHOULD NOT BE ZERO**

When we execute with correct values error will not occur in the program so after executing all the statements in the try block control directly jumps to the statements present after catch block without executing the catch block.

If any error occurs in the execution process in such case from the line where the error got occurred in the try block, control directly jumps to the catch block. So rest of the statement in try will not execute whereas catch block will execute.

**Note:**In our above program when the error got occurred we are displaying an error message “SECOND NUMBER SHOULD NOT BE ZERO”. In place of that error message we can also display the original error message associated with that data by calling a function “Error\_Message”. To test this rewriting the code inside the catch block as following

**Print Error\_Message()**

**Example 2: Create a procedure for dividing two variables values by using try catch implementation with system defined error statements.**

CREATE PROCEDURE SPDIVIDE**(**@X INT, @Y INT**)**

AS

BEGIN

DECLARE @Z INT

BEGIN TRY

SET @Z = @X / @Y

PRINT 'RESULT IS :-' + CAST**(**@Z ASCHAR**)**

END TRY

BEGIN CATCH

PRINT ERROR\_MESSAGE**()**

END CATCH

END

**EXEC SPDIVIDE 10, 0**  
**OUTPUT: Divide by zero error encountered.**

ADVERTISEMENT

**What is ERROR\_MESSAGE()?**

This method is used to display what type of error has occurred in the try block.

**PREDEFINED ERRORS:**

Whenever an error occurs under a program like dividing a number by zero, violation of primary key, violation of Check constraint etc, the system displays an error message telling us the problem encountered in the code.

Every error that occurs in the program is associated with four attributes

1. Error Number
2. Error Message
3. SEVERITY Level
4. State

**Example: Message 8134 (Error Number), Level 16(SEVERITY Level), State 1 (state), Divide by Zero error encountered (Error Message)**

**ERROR NUMBER:**Error number is a unique identification given for each and every error that occurs in the program. This value will be below 50,000 for predefined errors and must be above 50,000 for error defined by the user.

**ERROR MESSAGE:** It is a brief information describing the error occurred which should be maxed from 2047 characters.

**SEVERITY LEVEL:** This tells about the importance of the error which can be ranging between 0 to 24. In which  
**0 to 9:**are not serves which can be considered as information or status messages.  
**11 to 16:** Indicates these errors can be created by the user.  
**17 to 19:** Indicates these are software errors cannot be corrected by the user must be reported to the system administrator.  
**20 to 24:**Indicates fatal errors and if these errors occur they can damage the system or database. So here the connection immediately terminates with the database.

**STATE:**It is an arbitrary value which is not that important can be ranging from 0 to 127. We use this whenever the same error has to occur in multiple places.

**Note:**We can find the information of all predefined errors under the table “Sys Messages”

**How to raise errors explicitly in a program?**

Generally, errors raised in a program on predefined reasons like dividing a number by zero, violation of primary key, violation of check, violation of referential integrity etc. Whereas if required we can also raise an error in our programs in two different ways

1. Using Raiserror statement
2. Using a throw statement (new feature of SQL Server 2012)

**Syntax: Raiserror**  
**Raiserror (errorid/errormsg, SEVERITY, state)[with log]**

**Syntax: throw**  
**Throw errorid, errormsg, state**

**What is the difference between the RAISERROR function and throw statement?**

If we use any of the two statements in a program for raising an error without try and catch blocks, **RAISERROR**statement after raising the error will still continue the execution of the program whereas throw statement will terminate the program abnormally on that line. But if they are used under try block both will behave in the same way that is will jump directly to catch block from where the error got raised.

**RAISERROR**statement will give an option of specifying the SEVERITY of the error message whereas we don’t have these option in case of throw statement where all error message will have a default SEVERITY of 16.

In case of **RAISERROR**, there is a chance of recording the error message into the server log file by using the with log option whereas we cannot do this in case of throw statement.

In case of throw, we need to specify both errorid and error message to raise the error whereas in case of **RAISERROR** we can specify either id or message. If the id is not specified default error id is 50000 but if we want to specify only error id first we need to add the error message in the sysmessage table by specifying a unique id to the table.

**Example: Write a Procedure for dividing two numbers and raise an error in the program if the division is 1 by using the RAISERROR statement.**

CREATE PROCEDURE DIVIDEBYONE**(**@X INT, @Y INT**)**

AS

BEGIN

DECLARE @Z INT

SET @Z = 0

BEGIN TRY

IF @Y = 1

RAISERROR **(**'DIVISOR CANNOT BE ONE', 16, 1**)**

SET @Z = @X / @Y

PRINT'THE RESULT IS: '+CAST**(**@Z AS VARCHAR**)**

END TRY

BEGIN CATCH

PRINT ERROR\_NUMBER**()**

PRINT ERROR\_MESSAGE**()**

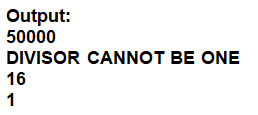
PRINT ERROR\_SEVERITY**()**

PRINT ERROR\_STATE**()**

END CATCH

END

**EXEC DIVIDEBYONE 100, 1**



The above procedure can also be defined with the help of a throw statement in place of Raiserror as following.

ALTER PROCEDURE DIVIDEBYONE**(**@X INT, @Y INT**)**

AS

BEGIN

DECLARE @Z INT

SET @Z = 0

BEGIN TRY

IF @Y = 1

THROW 50000,'DIVISOR CANNOT BE ONE', 1

SET @Z = @X / @Y

PRINT'THE RESULT IS: '+CAST**(**@Z AS VARCHAR**)**

END TRY

BEGIN CATCH

PRINT ERROR\_NUMBER**()**

PRINT ERROR\_MESSAGE**()**

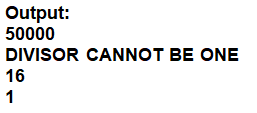
PRINT ERROR\_SEVERITY**()**

PRINT ERROR\_STATE**()**

END CATCH

END

**EXEC DIVIDEBYONE 100, 1**



**What is @@ERROR?**

The @@ERROR automatic variable returns the error code of the last Transact-SQL statement. If there was no error, @@ERROR returns zero. Because @@ERROR is reset after each Transact-SQL statement, it must be saved to a variable if it is needed to process it further after checking it.

Stored procedures report errors to client applications via the RAISERROR command. RAISERROR doesn’t change the flow of a procedure; it merely displays an error message, sets the @@ERROR automatic variable, and optionally writes the message to the SQL Server error log and the NT application event log.

**How to get @@error and @@rowcount at the same time?**

If @@Rowcount is checked after Error checking statement then it will have 0 as the value of @@Recordcount as it would have been reset.

And if @@Recordcount is checked before the error-checking statement then @@Error would get reset. To get @@error and @@rowcount at the same time do both in the same statement and store them in the local variable. SELECT @RC = @@ROWCOUNT, @ER = @@ERROR

# **SQL Server Stored Procedure Interview Questions**

**What is SubProgram?**

A subprogram in SQL Server is a named block of code that is directly saved on the database server. Then we can execute this subprogram when and where it is required. Under the databases, a subprogram is reported as a “stored procedure” or “stored function” or “database triggers”.

**What is a stored procedure or procedure in SQL Server?**

In a database management system (DBMS), a stored procedure is a precompiled set of Structured Query Language (SQL) statements with an assigned name and directly saved in the database.

The stored procedure in SQL Server can accept both input and output parameters so that a single stored procedure can be used by several clients over the network by using different input data. The stored procedure will reduce network traffic and increase performance. If we modify the body of the stored procedure then all the clients who are using the stored procedure will get the updated stored procedure.

**Why we need the stored procedure?**

This is one of the most frequently asked SQL Server Stored Procedure Interview Questions in the SQL Server Interview. So let’s discuss this question in details

Whenever we want to execute a SQL query from an application the SQL query (statement) what we send from an application will be first compiled(parsed) for execution, where the process of compiling(parsing) is time-consuming because compilation occurs each time when we execute the query or statements.

To overcome the above problem, we write SQL statements or query under the stored procedure and execute because a stored procedure is a pre-compiled block of code, without compiling(parsing) the statements get executed whenever the procedures are called which can increase the performance of database server which ultimately increases the performance of the application.

If we have a situation where we write the same query again and again, we can save that specific query as a stored procedure and call it’s just by its name whenever required that query to execute.

**What is the output parameter?**

The Parameters of a stored procedure can be of two types

1. Input parameters
2. Output parameters

The Input parameters are basically used for bringing value into the procedure for execution whereas the output parameters are used for carrying a value out of the procedure after execution.

If a parameter is declared as output, we only require assigning a value to the parameter inside the procedure so that the procedure will send that value at the end of procedure execution.

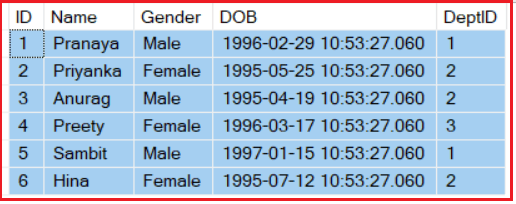
**What are the stored procedure status variables?**

Whenever we execute a stored procedure in SQL Server, it always returns an integer status variable indicating the status, usually, zero indicates success, and non-zero indicates the failure. To see this yourself, execute any stored procedure from the object explorer, in SQL Server management studio.

1. Right Click and select ‘Execute Stored Procedure
2. If the procedure, expects parameters, provide the values and click OK
3. Along with the result that you expect, the stored procedure also returns a Return Value = 0

So, from this point, we understood that, when a stored procedure is executed, it returns an integer status variable. With this in mind, let’s understand the difference between the output parameters and RETURN values in SQL Server stored procedure.

We are going to use the following Employee table to understand the Stored Procedure Output Parameters and Return values in SQL Server.



The following procedure returns the total number of employees from the Employee table, using the output parameter i.e. **@TotalCount**.

Create Procedure spGetTotalCountOfEmployees1

@TotalCount **int** output

As

Begin

Select @TotalCount = COUNT**(**ID**)**

From tblEmployee

End

Executing spGetTotalCountOfEmployees1 returns 6.

**Declare @TotalEmployees int**  
**Execute spGetTotalCountOfEmployees @TotalEmployees Output**  
**Select @TotalEmployees**

**Re-written stored procedure using return variables**

Create Procedure spGetTotalCountOfEmployees2

As

Begin

**return** **(**Select COUNT**(**ID**)**

From Employees**)**

End

Executing spGetTotalCountOfEmployees2 returns 6.

**Declare @TotalEmployees int**  
**Execute @TotalEmployees = spGetTotalCountOfEmployees2**  
**Select @TotalEmployees**

So, we are able to achieve what we want using output parameters as well as return values. Now let’s look at an example where return status variables cannot be used, but Output parameters can be used.

In this stored procedure, we are retrieving the name of the employee based on their Id using the output parameter i.e. **@Name**.

Create Procedure spGetNameById1

@Id **int**,

@Name nvarchar**(**20**)** Output

As

Begin

Select @Name = Name

From tblEmployee

Where Id = @Id

End

Executing the spGetNameById1 procedure will print the name of the employee

**Declare @EmployeeName nvarchar(20)**  
**Execute spGetNameById1 3, @EmployeeName out**  
**Print ‘Employee name:  ‘ + @EmployeeName**

**Now let’s try to do the same thing using the return status variables.**

Create Procedure spGetNameById2

@Id **int**

As

Begin

Return **(**Select Name

From tblEmployee

Where Id = @Id**)**

End

When we execute the **spGetNameById2** stored procedure, it will return an error stating **‘Conversion failed when converting the nvarchar value ‘Sam’ to data type int.’**. The return status variable is an integer, and hence when we select the name of an employee and try to return that we get the above conversion error.

**Declare @EmployeeName nvarchar(20)**  
**Execute @EmployeeName = spGetNameById2 1**  
**Print ‘Employee name: ‘ + @EmployeeName**

So using return values we can only return integers and that too, only one integer. It is not possible, to return more than one value using return values, whereas output parameters, can return any data type and a stored procedure can have more than one output parameter.

In general, RETURN values are used to indicate the success or failure of the stored procedure, especially when we are dealing with nested stored procedures. The return value of 0, indicates success, and any nonzero value indicates failure.

**Explain the Disadvantages of Return status value.**

1. We cannot return more than one value using the return status variable in SQL Server.
2. It is not possible to return values other than an integer using the return status variable in SQL Server Stored Procedure.

We can overcome the above two problems using the output variables in SQL Server Stored Procedure.

**What are the advantages of using a stored procedure in SQL Server?**

This SQL Server Stored Procedure Interview Questions asked in almost all interviews. The following are the advantages of using Stored Procedures over inline SQL queries in SQL Server.

**Execution Plan Retention and Reusability**

As there is no unnecessary compilation of queries this will reduces the burden on database (when we send a query to a SQL Server three things happen in order, 1st it checks the syntax of that query, 2ndit compiles that query, 3rd it generates an execution plan) as response User will get a quick response. Let’s get into more details.

The Stored Procedures are pre-compiled and their execution plan is cached and used again when the same stored procedure is executed again. Although ad-hoc queries also create and reuse plan, the plan is reused only when the query is the textual match and the datatypes are matching with the previous call. Any changes in the datatype or you have an extra space in the query then, a new plan is created.

**Reduces the Network Traffic**

The Stored Procedure reduces network traffic. When we execute a stored procedure we need to send the procedure name and parameters so only these things are passed on the network but if we are not using the stored procedure then we need to write the ad-hoc queries and we need to execute them which may contain many numbers of lines. So the stored procedure reduces the network traffic as a result performance of the application increase.

**Code Reusability and Better Maintainability**

Multiple applications can use the same stored procedure. The different applications which want similar kind of data then they can use the same stored procedure. The advantage is that if we want to change the stored procedure then we need to change it in one place that will affect to all the application that uses it whereas if it is inline SQL query and if we have to use it in multiple applications, then we end up with multiple copies of the same inline SQL query, and if the logic has to change, then we have to change the logic at all the places, which makes it harder maintaining inline SQL. So, the stored procedure provides code reusability and maintainability.

**Better Security**

By granting permission to the underlying database the user can do everything. He can view all the records as well as he can also modify the records. But if we want to restrict the user only to view the records then we need to grant only for that stored procedure which will display the records. In that way, we achieve better security with a stored procedure.

Using a stored procedure we can also avoid the SQL Injection attack.

**What is an execution plan?**

An execution plan is nothing but for the query to execute, what are the best possible ways available and it will choose the best possible way based on the indexes that available on the SQL Server to help that query. Based on those it generates the execution plan and then it executes the query. It will increase the performance of the application.

**What is an execution plan? When would you use it? How would you view the execution plan?**

An execution plan is basically a roadmap that graphically or textually shows the data retrieval methods chosen by the SQL Server query optimizer for a stored procedure or ad-hoc query and is a very useful tool for a developer to understand the performance characteristics of a query or stored procedure since the plan is the one that SQL Server will place in its cache and use to execute the stored procedure or query. From within Query Analyser is an option called “Show Execution Plan” (located on the Query drop-down menu). If this option is turned on it will display the query execution plan in the separate window when the query is run again.

**How to view the text of the stored procedure?**

Use system stored procedure sp\_helptext <user defined stored procedure name>

Ex: **sp\_helptext spGetEmployeeByGenderAndDepartment**

Right-click on the stored procedure in object explorer => Script procedure as =>Create To => new query editor window

**How to encrypt the text of a stored procedure?**

To encrypt the text of a stored procedure we use WITH ENCRYPTION option. For example

CREATE PROCEDURE SPWELCOME

WITH ENCRYPTION

AS

BEGIN

PRINT 'WELCOME TO PROCEDURE'

END

Now it is not possible to view the text of this encrypted procedure.

**What are the different procedure attributes in SQL Server?**

This is one of the frequently asked SQL Server Stored Procedure Interview Questions. There are two types of attributes

1. The With Encryption
2. With Recompile

**With Encryption Attribute:**

If this attribute is used on the procedure the text of this procedure is encrypted and will not be shown in the text column of the syscomments table so no one will be having an option to view the content of it.

**Note**: When an application is developed for a client at the time of installing this application on the client system we will be using the encryption option on all the views, procedures, functions, triggers, etc. and install on the client machine. So that they will not have the chance of viewing the source code or altering the source code.

**With Recompiled Attribute:**

1. Whenever a procedure is compiled for the first time it prepares the best query plan according to the current state of the database and executes the query plan when the procedure is called.
2. The compilation of the procedure and preparing a query plan is prepared not only at the time of procedure creation but each and every time the server is restarted (Implicitly occurs).
3. If the procedure is created by using with Recompile procedure attribute, it is forced to be compiled each time it is executed and whenever it compiles it prepares the query plan.
4. Forcing a procedure for recompilation and prepared a query plan is required when the database undergoes significant changes to its data or structure.
5. Another reason to force a procedure to recompile is if at all the tables is added with new indexes from which the procedure might be benefited forcing for recompilation is very important because we cannot wait until the server is restarted for preparing a new query plan.

**Note:** Even if the With Recompile option is available it is not suggested to be used if at all there are no significant changes in the structure of the databases.

**What is the temporary stored procedure?**

The stored procedures which are created temporarily in a database (not stored permanently) is called as the temporary stored procedure.

There are two types of temporary stored procedures such as

1. Private / Local Temporary Stored Procedure
2. Public / Global Temporary Stored Procedure.

**What are Private / Local Temporary Stored Procedure?**

1. These are created with the # prefix added to the procedure name.
2. Private or Local Temporary Stored Procedures are executed by the connection that created it.
3. These are automatically deleted when the connection created is closed.

**Syntax:**The syntax for creating a local temporary procedure

CREATE PROCEDURE #**<**PROCEDURE NAME**>**

AS

BEGIN

**<**PROCEDURE BODY / STATEMENTS / QUERIES**>**

END

**Example: Create a Local Temporary stored procedure.**

CREATE PROCEDURE #DelRec

AS

BEGIN

Delete **from** EMPLOYEE **where** Eid = 105

END

This procedure is executed on the session which is created it and once the session is closed this procedure is automatically deleted. And we cannot access this procedure once the session is closed also we cannot access this procedure from another session.

**What is the Public / Global Temporary Stored Procedure?**

1. These are created with the ## prefix added to the procedure name.
2. Any connection can execute the global temporary stored procedure.
3. A Global Temporary Stored Procedure exists until the connection used by the user who created the procedure is closed and any currently executing versions of the procedure by any other connections are implemented.
4. Once the connection that was used to create the procedure is closed, no further execution of the Global Temporary Stored Procedure is allowed. Only those connections that have already started executing the stored procedure are allowed to complete.

**Syntax:**The syntax for creating a Global Temporary Procedure

CREATE PROCEDURE ##**<**PROCEDURE NAME**>**

AS

BEGIN

**<**PROCEDURE BODY / STATEMENTS / QUERIES**>**

END

**Example: Create a Local Temporary stored procedure.**

CREATE PROCEDURE ##DelRec

AS

BEGIN

Delete **from** EMPLOYEE **where** Eid = 105

END

**What is the use of a Temporary Stored Procedure?**

Temporary Stored Procedures are useful when connecting to earlier versions of SQL Server that do not support the reuse of execution plans for Transact-SQL statements or batches.

**Example: Procedure with a default value.**

CREATE PROCEDURE PROC3**(**@X INT= 100, @Y INT**)**

AS

BEGIN

DECLARE @Z INT

SET @Z=@X+@Y

PRINT'The SUM of the 2 Numbers **is**: '+CAST**(**@Z AS VARCHAR**)**

END

**Executing the above procedure:**

**1. EXEC PROC3 200, 25**  
**2. EXEC PROC3 @X=200, @Y=25**  
**3. EXEC PROC3 @X=DEFAULT, @Y=25**  
**4. EXEC PROC3 @Y=25**

In the 3rd and 4th case, it uses the default value of 100 to the variable X which has been given while creating the procedure.

**What is deferred name resolution in SQL Server?**

This is one of the most frequently asked SQL Server Stored Procedure Interview Question in SQL Server Interview. Let me explain deferred name resolution with an example. Consider the stored procedure shown below.

Create procedure spGetCustomers

As

Begin

Select \* **from** Customers

End

**Customers** table does not exist. When we execute the above SQL code, the stored procedure **spGetCustomers** will be successfully created without errors. But when you try to **call or execute** the stored procedure using **Execute** **spGetCustomers**, we will get a runtime error stating Invalid object name ‘Customers’.

So, at the time of creating stored procedures, only the syntax of the SQL code is checked. The objects used in the stored procedure are not checked for their existence. Only when we try to run the procedure, the existence of the objects is checked. So, the process of postponing, the checking of the physical existence of the objects until runtime, is called deferred name resolution in the SQL Server.

**Functions** in SQL server does not support **deferred name resolution**. If we try to create an inline table-valued function as shown below, we get an error stating Invalid object name ‘Customers’ at the time of the creation of the function itself.

Create function fnGetCustomers**()**

returns table

**as**

**return** Select \* **from** Customers

So, this proves that stored procedures support deferred name resolution, whereas function does not. In fact, this is one of the major differences between functions and stored procedures in SQL Server.

**Can a stored procedure call itself or recursive stored procedure? How many level SP nesting possible?**

Yes. Because Transact-SQL supports recursion, you can write stored procedures that call themselves. Recursion can be defined as a method of problem solving wherein the solution is arrived at by repetitively applying it to subsets of the problem. A common application of recursive logic is to perform numeric computations that lend themselves to repetitive evaluation by the same processing steps. Stored procedures are nested when one stored procedure calls another or executes managed code by referencing a CLR routine, type, or aggregate. You can nest stored procedures and managed code references up to 32 levels.

# **SQL Server Joins Interview Questions**

**What are Joins in SQL Server?**

Joins in SQL Server are used to retrieve data from 2 more related tables. In general, tables are related to each other using foreign key constraints. An SQL JOIN is used to combine rows from two or more tables based on a common field between them.

**Explain the different types of Joins in SQL Server.**

JOINS are classified into two types such as

1. ANSI format JOINS
2. NON-ANSI format JOINS

Again the ANSI format joins classified into three types such as

1. Inner join
2. Outer join
3. Cross join

Further, the outer join divided into three types they are as follows

1. Left outer join
2. Right outer join
3. Full outer join

NON-ANSI format JOINS are classified into four types such as

1. EQUI join
2. NON-EQUI join
3. SELF-join
4. Natural Join

**What are the different types of joins available in SQL server?**

There are 3 different types of joins available in SQL server, and they are

1. Cross Join
2. Inner Join or Join
3. Outer Join

**What are ANSI and NON-ANSI Joins in SQL Server?**

When we retrieve the data from multiple tables based on ‘**ON**’ keyword condition then it is called ANSI format joins whereas when we retrieve the information from multiple tables based on ‘**WHERE**’ keyword condition then it is called as NON-ANSI format joins.

The most important point we need to keep in mind is that when we implement the JOIN mechanism on the tables then the tables should contain a related column.

When we execute the JOINS on the tables then each record of a table will join with each record of another table. JOINS are using for retrieving the data purpose only but not executed purpose. When we get the result from multiple tables, the result is temporary only but not permanent storage of data.

**What is the default JOIN in SQL Server?**

The default join of joins is INNER JOIN.

**What is the default join of outer join?**

The default join of the outer join is a full outer join.

**Explain about EQUI Join in SQL Server:**

When we retrieve the data from the table based on equality condition then it is known as EQUI join. (Use equal “=” operator only). It is NON-ANSI format join. Hence when we make a query for join using an equality operator, then that join query comes under EQUI join.

**Note:**  Output depends on the table order, not on the query condition.

**Explain about INNER JOIN in SQL Server.**

Inner join returns only the matching rows between both the tables. Non-matching rows are eliminated. The inner join is used to retrieve matching data from both the tables. The join that displays only the rows that have a match in both the joined tables is known as an inner join.

**Explain about OUTER JOIN in SQL Server.**

The OUTER JOIN is an extension of INNER JOIN or EQUI JOIN. In an inner join, the user will get matching data only from the tables i.e. the user will lose un-matching data from the tables. So to overcome the above problem we use the outer join mechanism.

By using the outer join we can retrieve matching data and also un-matching data from the tables. Outer join again classified into three types.

1. **Left outer join**
2. **Right outer join**
3. **Full outer join**

**What is the left outer join?**

It retrieves matching data from both the tables and also un-matching data from the left side table only. Un-matching data take null value in this case.

**What is the right outer join?**

It retrieves matching data from both the tables and also un-matching data from the right side table.

**What is the full outer join?**

It retrieves matching data and also un-matching data from both tables at a time.

**Explain about CROSS join in SQL Server.**

If two or more tables are combined with each other without any condition we call it a Cartesian or cross join. When we join the records without any condition is known as CROSS JOIN. In cross join, each record of a table is joins with each record of another table. Cross Join should not have an ON clause.

A cross joins that produces the Cartesian product of the tables that are involved in the join. The size of a Cartesian product is the number of rows in the first table multiplied by the number of rows in the second table like this.

**Explain about NON-EQUI JOIN in SQL Server.**

When we retrieve the information from the tables with any condition except equality condition then it is known as NON-EQUI JOIN. We may use <, >, <=,>=, <!,>!, AND, OR, NOT, etc except “=” operator.

**What is self-join in SQL Server?**

Joining a table by itself is called a self-join. When we have some relation between the columns within the same table then we use the self-join mechanism. When we implement a self-join mechanism we should create the alias for the table. We can create any number of aliases for a single table. Self-join is not a different kind of join. It can be classified of any type of join

1. **Inner Join**
2. **Outer (Left, Right, full) join**
3. **Cross Join**

Self-join is just like any other join except that two instances of the same table will be joined in the query. Here is an example: Employees table which contains rows for normal employees as well as managers of that employee. So to find out the managers of all the employees, we need a self-join.

**Differences between UNION and UNION ALL in SQL Server:**

Both UNION and UNION ALL are used to combines the result-set of two or more select queries into a single result-set. The difference between these two is UNION removes duplicate rows whereas UNION ALL does not. When use UNION, to remove the duplicate rows, the SQL server has to do a distinct sort which is time-consuming. For this reason, the UNION ALL is much faster than UNION.

If we want to see the cost of DISTINCT SORT, you can turn on the estimated query execution plan using CTRL + L. For UNION and UNION ALL to work, the Number, Data types, and the order of the columns in the select statements should be the same.

If we want to sort the results of UNION or UNION ALL the ORDER BY clause should be used on the last SELECT statement as shown below.

**Difference between JOIN and UNION in SQL Server:**

JOINS and UNIONS are two different things. However, this question is being asked very frequently now. UNION combines the result-set of two or more select queries into a single result-set whereas JOINS retrieves data from two or more tables based on logical relationships between the tables.

**What is a cross join? Explain with an example?**

Let us understand Cross Join with an example. Create 2 tables Company and Candidate. Use the script below to create these tables and populate them. CompanyId column in Candidate Table is a foreign key referencing CompanyId in Company Table.

CREATE TABLE Company

**(**

CompanyId TinyInt Identity Primary Key,

CompanyName Nvarchar**(**50**)** **NULL**

**)**

GO

INSERT Company VALUES**(**'DELL'**)**

INSERT Company VALUES**(**'HP'**)**

INSERT Company VALUES**(**'IBM'**)**

INSERT Company VALUES**(**'Microsoft'**)**

GO

CREATE TABLE Candidate

**(**

CandidateId tinyint identity primary key,

FullName nvarchar**(**50**)** **NULL**,

CompanyId tinyint REFERENCES Company**(**CompanyId**)**

**)**

GO

INSERT Candidate VALUES**(**'Ron',1**)**

INSERT Candidate VALUES**(**'Pete',2**)**

INSERT Candidate VALUES**(**'Steve',3**)**

INSERT Candidate VALUES**(**'Steve',**NULL)**

INSERT Candidate VALUES**(**'Ravi',1**)**

INSERT Candidate VALUES**(**'Raj',3**)**

INSERT Candidate VALUES**(**'Kiran',**NULL)**

GO

A cross join produces the Cartesian product of the tables involved in the join. The size of a Cartesian product result set is the number of rows in the first table multiplied by the number of rows in the second table. A query involving a CROSS JOIN for the Candidate and Company Table is shown below.

SELECT Cand.CandidateId,

Cand.FullName,

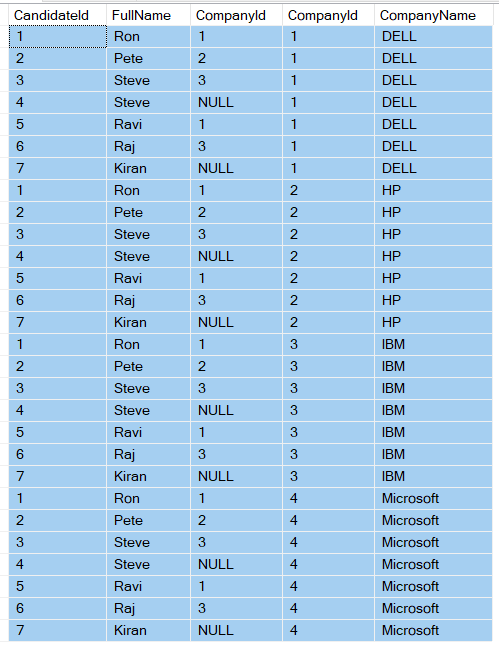
Cand.CompanyId,

Comp.CompanyId,Comp.CompanyName

FROM Candidate Cand

CROSS JOIN Company Comp

**If we run the above query, we produce the result set shown in the image below.**



**Key Points to remember about CROSS JOIN.**

A cross join produces the Cartesian product of the tables involved in the join. This means every row in the Left Table is joined to every row in the Right Table. The candidate is LEFT Table and Company is RIGHT Table. In our example, we have 28 total numbers of rows in the result set. 7 rows in the Candidate table multiplied by 4 rows in the Company Table.

In real-time scenarios, we rarely use a CROSS JOIN. Most often we use either INNER JOIN or LEFT OUTER JOIN. The CROSS JOIN does not have an ON clause with Join’s conditions. All the other JOINS use the ON clause with a Join Condition. Using an ON clause on a CROSS JOIN would generate a syntax error.

**What is inner Join in SQL Server? Explain with an example?**

Let us understand the **Inner join**with an example. Create 2 Tables **Company** and **Candidate**. We are going to use the following two tables through out this article. Please use the script below to create these tables and populate them. CompanyId column in Candidate Table is a foreign key referencing CompanyId in Company Table.

CREATE TABLE Company

**(**

CompanyId TinyInt Identity Primary Key,

CompanyName Nvarchar**(**50**)** **NULL**

**)**

GO

INSERT Company VALUES**(**'DELL'**)**

INSERT Company VALUES**(**'HP'**)**

INSERT Company VALUES**(**'IBM'**)**

INSERT Company VALUES**(**'Microsoft'**)**

GO

CREATE TABLE Candidate

**(**

CandidateId tinyint identity primary key,

FullName nvarchar**(**50**)** **NULL**,

CompanyId tinyint REFERENCES Company**(**CompanyId**)**

**)**

GO

INSERT Candidate VALUES**(**'Ron',1**)**

INSERT Candidate VALUES**(**'Pete',2**)**

INSERT Candidate VALUES**(**'Steve',3**)**

INSERT Candidate VALUES**(**'Steve',**NULL)**

INSERT Candidate VALUES**(**'Ravi',1**)**

INSERT Candidate VALUES**(**'Raj',3**)**

INSERT Candidate VALUES**(**'Kiran',**NULL)**

GO

If we want to select all the rows from the LEFT table (In our example Candidate Table) that have a non-null foreign key value (CompanyId in Candidate Table is the foreign key) then we use INNER JOIN. **A query involving an INNER JOIN for the Candidate and Company Table is shown below.**

SELECT Cand.CandidateId,

Cand.FullName,

Cand.CompanyId,

Comp.CompanyId,

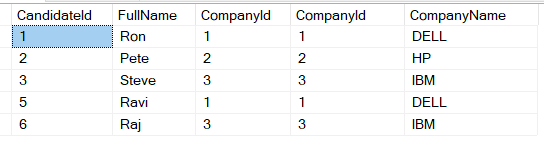
Comp.CompanyName

FROM Candidate Cand

INNER JOIN Company Comp

ON Cand.CompanyId = Comp.CompanyId

If we run the above query the output will be as shown in the image below. If we look at the output, we only got 5 rows. We did not get the 2 rows that have a NULL value in the CompanyId column. So, an INNER JOIN would get all the rows from the LEFT table that has non-null foreign key value.



Instead of using the INNER JOIN keyword we can just use the JOIN keyword as shown below. JOIN or INNER JOIN means the same.

SELECT Cand.CandidateId,

Cand.FullName,

Cand.CompanyId,

Comp.CompanyId,

Comp.CompanyName

FROM Candidate Cand

JOIN Company Comp

ON Cand.CompanyId = Comp.CompanyId

**What is the Left Outer Join in SQL Server? Explain with an example?**

Let us understand **Left join in SQL Server**with an example. If we want to select all the rows from the LEFT table (In our example Candidate Table) including the rows that have a null foreign key value (CompanyId in Candidate Table is the foreign key ) then we use LEFT OUTER JOIN. A query involving a LEFT OUTER JOIN for the Candidate and Company Table is shown below.

SELECT Cand.CandidateId,

Cand.FullName,

Cand.CompanyId,

Comp.CompanyId,

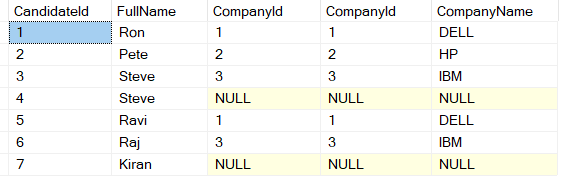
Comp.CompanyName

FROM Candidate Cand

LEFT OUTER JOIN Company Comp

ON Cand.CompanyId = Comp.CompanyId

If we run the above query the output will be as shown below. If we look at the output, we now got all 7 rows (all the rows from the Candidate Table) including the row that has a null value for the CompanyId column in the Candidate Table. So, the LEFT OUTER JOIN would get all the rows from the LEFT Table including the rows that have null foreign key value.



Instead of using the LEFT OUTER JOIN keyword we can just use the LEFT JOIN keyword as shown below. LEFT OUTER JOIN or LEFT JOIN means the same.

SELECT Cand.CandidateId,

Cand.FullName,

Cand.CompanyId,

Comp.CompanyId,

Comp.CompanyName

FROM Candidate Cand

LEFT JOIN Company Comp

ON Cand.CompanyId = Comp.CompanyId

**What is the Right Outer Join in SQL Server? Explain with an example?**

Let us understand Right Outer join in SQL Server with an example. If we want to select all the rows from the LEFT Table (In our example Candidate Table) that have non-null foreign key values plus all the rows from the RIGHT table (In our example Company Table) including the rows that are not referenced in the LEFT Table, then we use RIGHT OUTER JOIN. A query involving a RIGHT OUTER JOIN for the Candidate and Company Table is shown below.

SELECT Cand.CandidateId,

Cand.FullName,

Cand.CompanyId,

Comp.CompanyId,

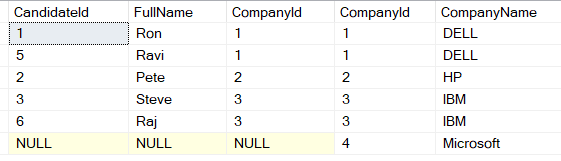
Comp.CompanyName

FROM Candidate Cand

RIGHT OUTER JOIN Company Comp

ON Cand.CompanyId = Comp.CompanyId

If we run the above query the output will be as shown below. If we look at the output, we now got 6 rows. All the rows from the Candidate Table that has non-null foreign key value plus all the rows from the Company Table including the row that is not referenced in the Candidate Table.



Instead of using the RIGHT OUTER JOIN keyword we can just use the RIGHT JOIN keyword as shown below. RIGHT OUTER JOIN or RIGHT JOIN means the same.

SELECT Cand.CandidateId,

Cand.FullName,

Cand.CompanyId,

Comp.CompanyId,

Comp.CompanyName

FROM Candidate Cand

RIGHT JOIN Company Comp

ON Cand.CompanyId = Comp.CompanyId

**What is Full Outer Join in SQL Server? Explain with an example?**

Let us understand Full Outer join in SQL Server with an example. If we want to select all the rows from the LEFT Table (In our example Candidate Table) plus all the rows from the RIGHT table (In our example Company Table), then we use FULL OUTER JOIN. A query involving a FULL OUTER JOIN for the Candidate and Company Table is shown below.

SELECT Cand.CandidateId,

Cand.FullName,

Cand.CompanyId,

Comp.CompanyId,

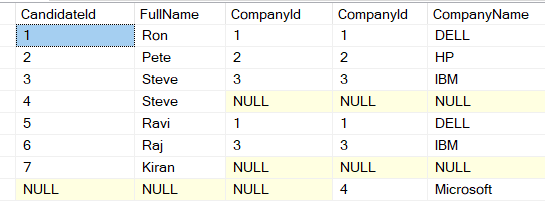
Comp.CompanyName

FROM Candidate Cand

FULL OUTER JOIN Company Comp

ON Cand.CompanyId = Comp.CompanyId

If we run the above query the output will be as shown below. If you look at the output, we now got 8 rows. All the rows from the Candidate Table and all the rows from the Company Table.



Instead of using FULL OUTER JOIN keyword we can just use FULL JOIN keyword as shown below. FULL OUTER JOIN or FULL JOIN means the same.

SELECT Cand.CandidateId,

Cand.FullName,

Cand.CompanyId,

Comp.CompanyId,

Comp.CompanyName

FROM Candidate Cand

FULL JOIN Company Comp

ON Cand.CompanyId = Comp.CompanyId