

SQL Server – Basics

- ✔ [Connecting to SQL Server using SQL Server Management Studio](#)
- ✔ [Creating Altering and Deleting Database in SQL Server](#)
- ✔ [Creating Altering and Deleting Tables in SQL server](#)
- ✔ [SQL Server Data Types](#)
- ✔ [Constraints in SQL Server](#)
- ✔ [Primary Key in SQL Server](#)
- ✔ [Foreign Key in SQL Server](#)
- ✔ [Primary Key and Foreign key Relationship Between Multiple Tables in SQL Server](#)
- ✔ [Cascading Referential Integrity Constraint in SQL Server](#)
- ✔ [Identity Column in SQL Server](#)
- ✔ [Sequence Object in SQL Server](#)
- ✔ [Difference Between Sequence and Identity in SQL Server](#)
- ✔ [Select Statement in SQL Server](#)

SQL Server – Clauses

- ✔ [Where Clause in SQL Server](#)
- ✔ [Order By Clause in SQL Server](#)
- ✔ [Top n Clause in SQL Server](#)
- ✔ [Group By Clause in SQL Server](#)
- ✔ [Having Clause in SQL Server](#)
- ✔ [Difference Between Where and Having Clause in SQL Server](#)

SQL Server – Operators

- ✔ [Assignment Operator in SQL Server](#)
- ✔ [Arithmetic Operators in SQL Server](#)
- ✔ [Comparison Operators in SQL Server](#)
- ✔ [Logical Operators in SQL Server](#)
- ✔ [IN BETWEEN and LIKE Operators in SQL Server](#)
- ✔ [ALL Operator in SQL Server](#)
- ✔ [ANY Operator in SQL Server](#)
- ✔ [SOME Operator in SQL Server](#)
- ✔ [EXISTS Operator in SQL Server](#)
- ✔ [UNION and UNION ALL Operators in SQL Server](#)
- ✔ [EXCEPT Operator in SQL Server](#)
- ✔ [INTERSECT Operator in SQL Server](#)
- ✔ [Differences Between UNION EXCEPT and INTERSECT Operators in SQL Server](#)

SQL Server – JOINS

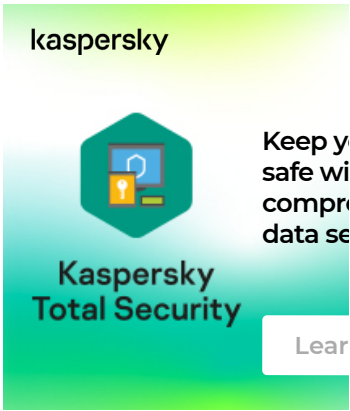
- ✔ [Joins in SQL Server](#)
- ✔ [Inner Join in SQL Server](#)
- ✔ [Left Outer Join in SQL Server](#)
- ✔ [Right Outer Join in SQL Server](#)
- ✔ [Full Outer Join in SQL Server](#)
- ✔ [SQL Server Self Join](#)
- ✔ [Cross Join in SQL Server](#)

SQL Server – Indexes

- ✔ [Indexes in SQL Server](#)
- ✔ [Clusted Index in SQL Server](#)

ACID Properties in SQL Server

Back to: [SQL Server Tutorial For Beginners and Professionals](#)



ACID Properties in SQL Server with Example

In this article, I am going to discuss the **ACID Properties in SQL Server** with examples. Please read our previous article before proceeding to this article where we discussed [Nested Transactions in SQL Server](#) with examples. As part of this article, we are going to discuss the following pointers.

1. [What is a Transaction in SQL Server?](#)
2. [What are ACID Properties of a Transaction?](#)
3. [Understanding the ACID Properties in SQL Server](#)
4. [Atomicity Property of a Transaction](#)
5. [Consistency Property of a Transaction in SQL Server](#)
6. [Isolation Property of a Transaction](#)
7. [Durability Property of a Transaction in SQL Server](#)

What is a Transaction in SQL Server?

A transaction in the SQL server is a group of SQL statements that are treated as a single unit and they are executed by applying the principle of “**do everything or do nothing**” and a successful transaction must pass the ACID test.

What are ACID Properties of a Transaction?

In the context of transaction processing, the acronym ACID refers to the four key properties of a transaction, such as

1. **Atomicity**
2. **Consistency**
3. **Isolation**
4. **Durability.**

Understanding the ACID Properties in SQL Server:

Let us understand the ACID Properties of a transaction in SQL Server. In order to understand this, here, we are going to use the following two tables.

ProductID	Name	Price	Quantity	ProductSalesId	Produc	QuantitySold
101	Laptop	15000	100	1	101	10
102	Desktop	20000	150	2	102	15
103	Mobile	3000	200	3	103	30
104	Tablet	4000	250	4	104	35

Product Table

ProductSales Table

Please use below SQL scripts to create and populate the Product and ProductSales table with the test data.

```
-- Create the Product Table
CREATE TABLE Product
(
    ProductID INT PRIMARY KEY,
    Name VARCHAR(40),
    Price INT,
    Quantity INT
)
GO

-- Populate the Product Table with some test data
INSERT INTO Product VALUES(101, 'Laptop', 15000, 100)
INSERT INTO Product VALUES(102, 'Desktop', 20000, 150)
```

- ✔ [Non-Clustered Index in SQL Server](#)
- ✔ [How Index impacts DML Operations](#)
- ✔ [SQL Server Unique Index](#)
- ✔ [Index in GROUP BY Clause in SQL Server](#)
- ✔ [Advantages and Disadvantages of Indexes in SQL Server](#)

SQL Server – Built-in Functions

- ✔ [Built-in String Functions in SQL Server](#)
- ✔ [OVER Clause in SQL Server](#)
- ✔ [Row_Number Function in SQL Server](#)
- ✔ [Rank and Dense_Rank Function in SQL Server](#)

User Defined Functions and Stored Procedure

- ✔ [Stored Procedure in SQL Server](#)
- ✔ [SQL Server Stored Procedure Return Value](#)
- ✔ [SQL Server Temporary Stored Procedure](#)
- ✔ [Stored Procedure with Encryption and Recompile Attribute](#)
- ✔ [Scalar Function in SQL Server](#)
- ✔ [Inline Table Valued Function in SQL Server](#)
- ✔ [Multi Statement Table Valued Function in SQL Server](#)
- ✔ [Encryption and Schema Binding Option in SQL Server Functions](#)
- ✔ [Deterministic and Non-Deterministic Functions in SQL Server](#)

Exception Handling and Transaction Management

- ✔ [Transaction Management in SQL Server](#)
- ✔ [Types of Transactions in SQL Server](#)
- ✔ [Nested Transactions in SQL Server](#)
- ✔ [ACID Properties in SQL Server](#)
- ✔ [Exception Handling in SQL Server](#)
- ✔ [RaiseError in SQL Server](#)
- ✔ [How to Raise Errors Explicitly in SQL Server](#)
- ✔ [Exception Handling Using Try Catch in SQL Server](#)

Views and Triggers in SQL Server

- ✔ [Views in SQL Server](#)
- ✔ [Advantages and Disadvantages of Views in SQL Server](#)
- ✔ [Complex Views in SQL Server](#)
- ✔ [Views with Check Encryption and Schema Binding Options in SQL Server](#)
- ✔ [Indexed View in SQL Server](#)
- ✔ [Triggers in SQL Server](#)
- ✔ [Inserted and Deleted Tables in SQL Server](#)
- ✔ [DML Trigger Real-Time Examples in SQL Server](#)
- ✔ [Instead Of Trigger in SQL Server](#)

```
INSERT INTO Product VALUES(103, 'Mobile', 3000, 200)
INSERT INTO Product VALUES(104, 'Tablet', 4000, 250)
```

```
-- Create the ProductSales table
CREATE TABLE ProductSales
(
    ProductSalesId INT PRIMARY KEY,
    ProductId INT,
    QuantitySold INT
)
GO
```

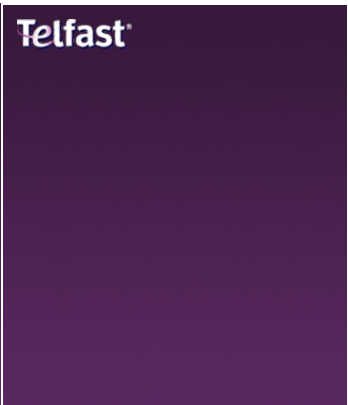
```
-- Populate the ProductSales table with some test data
INSERT INTO ProductSales VALUES(1, 101, 10)
INSERT INTO ProductSales VALUES(2, 102, 15)
INSERT INTO ProductSales VALUES(3, 103, 30)
INSERT INTO ProductSales VALUES(4, 104, 35)
GO
```

Let us discuss each of these properties of a transaction one by one with an example.

Atomicity Property of a Transaction in SQL Server:

The **Atomicity Property of a Transaction in SQL Server** ensures that either all the DML Statements (i.e. insert, update, delete) inside a transaction are completed successfully or all of them are rolled back.

For example, in the following, **spSellProduct** stored procedure, both the **UPDATE** and **INSERT** statements should succeed. If the **UPDATE** statement succeeds and the **INSERT** statement fails, the database should undo the changes made by the **UPDATE** statement, by rolling it back.



```
CREATE PROCEDURE spSellProduct
@ProductID INT,
@QuantityToSell INT
AS
BEGIN
    -- First we need to check the stock available for the product we want to sell
    DECLARE @StockAvailable INT

    SELECT @StockAvailable = Quantity
    FROM Product
    WHERE ProductId = @ProductId

    -- We need to throw an error to the calling application
    -- if the stock is less than the quantity we want to sell
    IF(@StockAvailable< @QuantityToSell)
    BEGIN
        Raiserror('Enough Stock is not available',16,1)
    END
    -- If enough stock is available
    ELSE
    BEGIN
        BEGIN TRY
            -- We need to start the transaction
            BEGIN TRANSACTION

            -- First we need to reduce the quantity available
            UPDATE Product SET
                Quantity = (Quantity - @QuantityToSell)
            WHERE ProductID = @ProductID

            -- Calculate MAX ProductSalesId
            DECLARE @MaxProductSalesId INT
            SELECT @MaxProductSalesId = CASE
                WHEN MAX(ProductSalesId) IS NULL THEN 0
                ELSE MAX(ProductSalesId)
            END
            FROM ProductSales

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

            -- We need to insert the quantity sold into the ProductSales table
            INSERT INTO ProductSales(ProductSalesId, ProductId, QuantitySold)
```

- DDL Triggers in SQL Server
- Triggers Execution Order in SQL Server
- Creating and Managing Users in SQL Server
- Logon Triggers in SQL Server

Concurrent Transactions and DeadLock in SQL Server

- Concurrency Problems in SQL Server
- Dirty Read Concurrency Problem in SQL Server
- Lost Update Concurrency Problem in SQL Server
- Non-Repeatable Read Concurrency Problem
- Phantom Read Problem in SQL Server
- Snapshot Transaction Isolation Level in SQL Server
- Read Committed Snapshot Isolation Level
- Difference between Snapshot Isolation and Read Committed Snapshot
- Deadlock in SQL Server
- Deadlock Victim Selection in SQL Server
- Deadlock Logging in SQL Server Error Log
- SQL Server Deadlock Analysis and Prevention
- Capturing Deadlocks using SQL Profiler
- SQL Server Deadlock Error Handling
- How to Find Blocking Queries in SQL Server

Advanced Concepts

- Database Normalization in SQL Server
- De-normalization in SQL Server
- Star Schema vs Snow Flake Design
- How to Schedule Jobs in SQL Server using SQL Server Agent
- How SQL Server Store and Manages Data Internally
- Change Data Capture in SQL Server
- How to Implement PIVOT and UNPIVOT in SQL Server
- Reverse PIVOT Table in SQL Server

Performance Improvements in SQL Server Query

- Performance Improvements in SQL Server
- Performance Improvement using Unique Keys
- When to Choose Table Scan and when to choose Seek Scan
- How to Use Covering Index to reduce RID lookup
- Create Index on Proper Column to Improve Performance
- Performance Improvement using Database Engine Tuning Advisor

```
VALUES(@MaxProductSalesId, @ProductId, @QuantityToSell)

-- Finally Commit the transaction
COMMIT TRANSACTION
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION
END CATCH
End
END
```

Consistency Property of a Transaction in SQL Server:

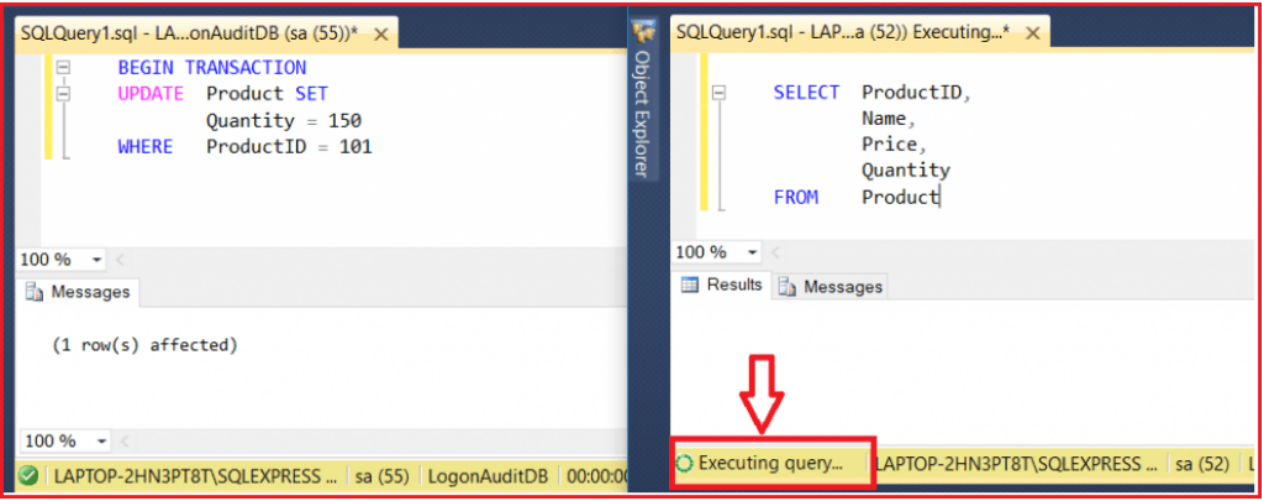
The **Consistency Property of a Transaction in SQL Server** ensures that the database data is in a consistent state before the transaction started and also left the data in a consistent state after the transaction completed. If the transaction violates the rules then it should be rolled back. For example, if stocks available are decremented from the **Product** table then there has to be a related entry in the **ProductSales** table.

In our example, let say, the transaction has updated the quantity available in the product table, and suddenly there is a system failure (right before the insertion into the ProductSales table or in the middle). In this situation the system will roll back the updates Otherwise, we can't trace out the stock information.

Isolation Property of a Transaction in SQL Server:

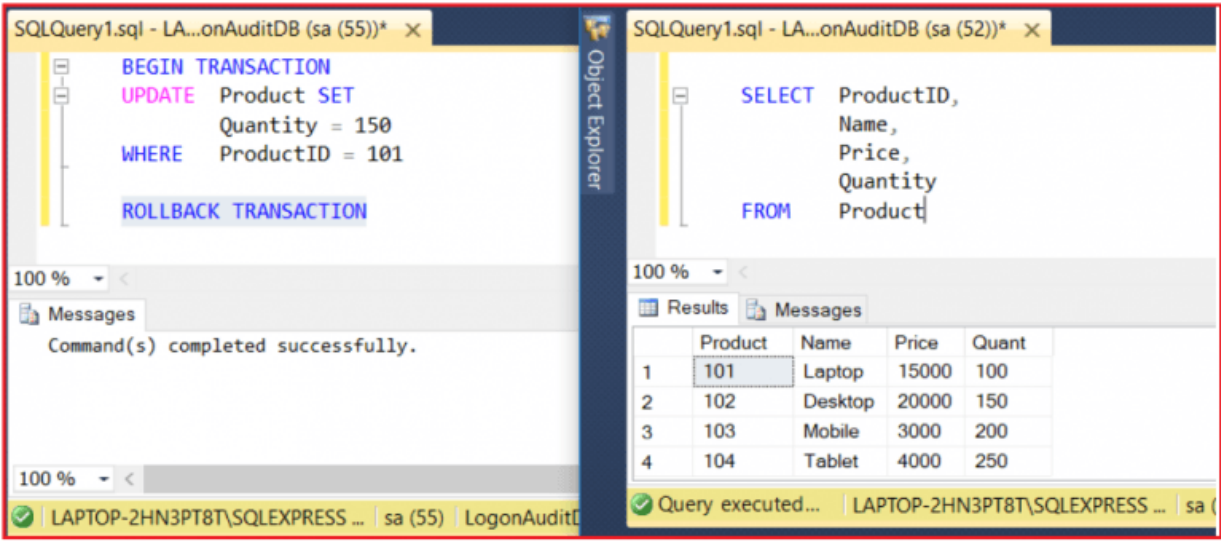
The **Isolation Property of a Transaction in SQL Server** ensures that the intermediate state of a transaction is invisible to other transactions. The Data modifications made by one transaction must be isolated from the data modifications made by all other transactions. Most databases use locking to maintain transaction isolation.

To understand the Isolation Property of a Transaction, we are going to use two separate instances of SQL Server. From the **First Instance**, we started the transaction and updating the record in the Product table but we haven't committed or rolled back the transaction. And from the **Second Instance**, we are using the select statement to select the records present in the Product table as shown in the below image.



As you can see from the above screenshot, the select statement is not returning the data. The reason is we can't access the intermediate state of a transaction by default. In a later article, we will discuss how to see the intermediate state of a transaction in SQL Server.

Let execute the Rollback transaction. This will immediately show the result of the Select statement because the lock is released from the Product table as shown in the below image.



Durability Property of a Transaction in SQL Server:

The **Durability Property of a Transaction in SQL Server** ensures that once the transaction is successfully completed, then the changes it made to the database will be permanent. Even if there is a system failure or power failure or any abnormal changes, it should safeguard the committed data.

Note: The acronym ACID was created by Andreas Reuter and Theo Härder in the year 1983, however, Jim Gray had already defined these properties in the late 1970s. Most of the popular databases such as SQL Server, Oracle, MySQL, Postgre SQL follows the ACID properties by default.



In the next article, I am going to discuss [Exception Handling in SQL Server](#) with examples. Here, in this article, I try to explain **ACID Properties in SQL Server** with Examples. I hope you enjoy this ACID Properties in SQL Server with Examples article.

DigitalOcean® Developer Cloud - Simple, Powerful Cloud Hosting

Ad try.digitalocean.com

Deadlock in SQL Server with Examples

dotnettutorials.net

أشهر المواقع العالمية تصل لبيتك

Ad Lynks.com

SQL Server Management Studio

dotnettutorials.net

ESL for F1 Visa Students

Ad UCEDA International

Performance Improvement using Unique Keys

dotnettutorials.net

Performance Improvements in SQL Server

dotnettutorials.net

Views in SQL ؟ with Examples

dotnettutorials.net

[← Previous Lesson](#) [Next Lesson →](#)
Nested Transactions in SQL Server **Exception Handling in SQL Server**

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

Website

Post Comment