DOT NET TUTORIALS

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
101	Laptop	15000	100
102	Desktop	20000	150
103	Mobile	3000	200
104	Tablet	4000	250

Product Table

ProductSalesId	Produc	QuantitySold
1	101	10
2	102	15
3	103	30
4	104	35

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)
```

```
 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)
```

```
      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:

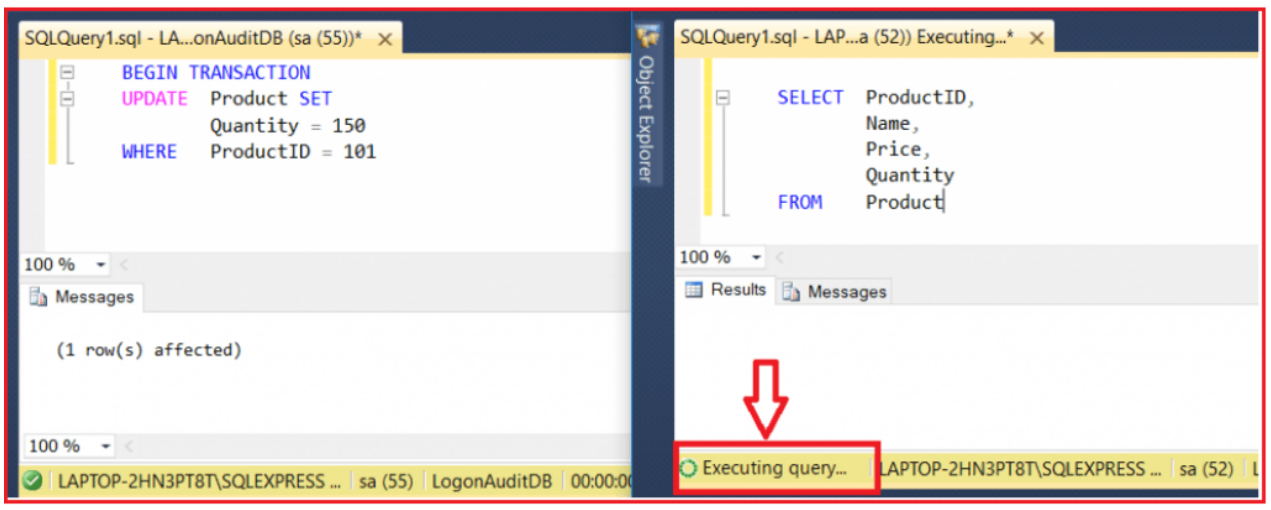
The **Consistency Property of a Transaction in SQL Server** ensures that the database data is in a consistent state before the transaction started and also left the data in a consistent state after the transaction completed. If the transaction violates the rules then it should be rolled back. For example, 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:

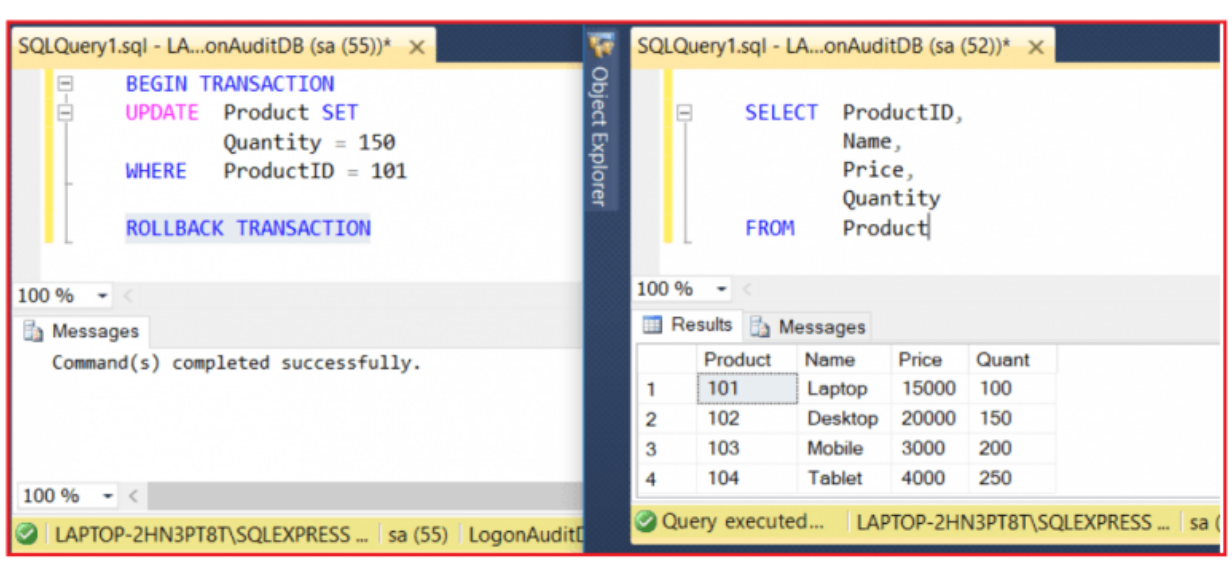
The **Isolation Property of a Transaction in SQL Server** ensures that the intermediate state of a transaction is invisible to other transactions. The Data modifications made by one transaction must be isolated from the data modifications made by all other transactions. Most databases use locking to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

← Previous Lesson
Nested Transactions in SQL Server

Next Lesson →
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