

Eugene Borts

Applied Database II

Dr. Ron Eaglin

Assignment 5

## Part One – The ACID Properties

ACID is an acronym that stands for a set of properties used to guarantee integrity in database transactions. The four properties comprising ACID are known as Atomicity, Consistency, Isolation, and Durability. Atomicity states that all or no transactions are committed. It means that transactions can only be executed in an all-or-nothing manner, either committing when all requirements are satisfied or not executing at all. There are other ways Atomicity can be perceived, but the primary two are completeness and mutual exclusion. Consistency states that if a transaction begins in a consistent state, then it will end in a consistent state, and if a transaction is not completed then it is returned to the previous consistent state. Isolation states that transactions must be isolated from each other, as the name would suggest, so that no transaction may interfere with any other transaction. Durability states that each completed transaction must be saved and stored securely. These four statements are known as the ACID properties, or the database transaction properties that make up the ACID rule set. This set of standards is intended to guarantee validity even in the event of errors, power failures, inconsistencies, and compromises.

A database operation that satisfies all four of the ACID properties is known as a transaction. Transactions are often composed of multiple statements, but in accordance with Atomicity each transaction is treated as a single unit. This unit either completely succeeds or completely fails on its own. If any statement in the unit fails to execute, then the entire transaction fails to commit, and the database remains unchanged. Atomicity states that multiple operations can be grouped into a single logical entity (Oracle). A series of database operations in an atomic transaction will either all occur, or none will occur. This series of operations cannot be separated and is indivisible by its very nature, which is the foundation of atomicity. In other words, atomicity is synonymous with indivisibility and irreducibility (Oracle).

Consistency functions by ensuring that each transaction can only alter the database from one valid state to another, maintaining database invariants. This ensures that any data written to the database must be validated according to all defined rules. Consistency may seem as though it guarantees that each transaction is correct, but in fact it only prevents illegal transactions and database corruption without guaranteeing that each transaction is correct. Consistency demands that all validation rules must be met for data to be processed, which ensures that each transaction is valid, but does not ensure that a transaction is correct. This is known as a validation check, and it is at the heart of upholding consistency. If a validation check shows that the data in a transaction is inconsistent with the

rules of the database, the entire transaction must be cancelled, and all rows must be returned to their previous state.

Concurrency control is a common process in the execution of transactions, as transactions are often executed in a concurrent manner. In-memory changes can be accessed by multiple concurrent transactions; thus, concurrency control mechanisms are employed to safeguard against such vulnerabilities and ensure data integrity (Mihalcea, 2019). An example of a concurrency control mechanism is the transaction log. When a transaction has modified a table row, the uncommitted changes are applied to the in-memory structures while the previous data is stored temporarily in the transaction log. One of the primary goals of concurrency control is isolation, which ensures that concurrent execution of transactions sets the database to the same state that would have been reached if the transactions were executed sequentially. By adhering to the standards of isolation, the effects of an incomplete transaction may not be visible to other transactions. An example of isolation is two transactions attempting to modify the same data at once. If one of the transactions remains idle until the other is completed, then the isolation property is satisfied. If a transaction is completely isolated, then it commits as the only task executing against the database during its lifetime (White, 2014).

Durability is perhaps the most straightforward ACID property, both in name and functionality. Living up to the standard of durability entails the recording of completed transactions, and their effects, in non-volatile memory. This guarantees that a transaction will remain committed once it has been completed, even in the case of a system failure. Simply put, durability is the storage and safekeeping of completed transactions. One method of establishing durability in database transactions is by using a redo log. A redo log is an append-only disk-based structure that stores every change a given transaction has undergone. This means that once a transaction commits, every data page will also be written in the redo log. Writing to the redo log is faster than random access due to sequential disk access and using a redo log allows for fast transactions in general, especially when compared to flushing the data pages. In SQL Server, the transaction log is used in place of the redo log (Mihalcea, 2019).

ACID plays a highly important role in database transactions which must be considered very carefully. A database management system that supports transactions must strive to support all ACID properties, and any commercial DBMS must provide full ACID support. Any database that can be accessed by multiple users should adhere to all ACID properties to ensure the integrity of its data (Will A.).

#### Works Cited

Oracle Documentation. Retrieved from  
[https://docs.oracle.com/cd/E17276\\_01/html/programmer\\_reference/transapp\\_atomicity.html](https://docs.oracle.com/cd/E17276_01/html/programmer_reference/transapp_atomicity.html)

White, P. (February 27, 2014). The ACID Properties of Statements & Transactions. Retrieved from <https://sqlperformance.com/2014/02/t-sql-queries/confusion-caused-by-trusting-acid>

Mihalcea, V. (January 22, 2019). How does a relational database work. Retrieved from <https://vladmihalcea.com/how-does-a-relational-database-work/>

Mihalcea, V. and A, Will. (April 1, 2018). Stack Overflow. Retrieved from <https://stackoverflow.com/questions/3740280/acid-and-database-transactions>

## Part Two – Transactions

The set of stored procedures below adheres to the ACID properties by validating that the specified accounts used in a transaction exist and determining whether the account balance is sufficient to perform the transaction. If the origin or target account does not exist, or if there is an insufficient balance in the account from which currency is being withdrawn, then the transaction will not be committed, and the database will be rolled back to its previous state. When a transaction is committed, the results are logged in the Transactions table using a transaction log. The transaction type is logged as a credit or a withdrawal using Boolean values. If the balance on an account transfer is negative for that account, then the transaction type is a 0 for credit. If a balance on an account transfer is negative for that account, then the transaction type is a 1 for withdrawal. When a transaction is not committed, an error message is logged detailing the issue that prevented that transaction from executing.

### Create Tables

```
USE UserCreditDB

CREATE TABLE Users (UserID int NOT NULL PRIMARY KEY, UserName varchar(50), UserAddress varchar(255));

CREATE TABLE UserAccount (UserAccountID int NOT NULL PRIMARY KEY, UserID int FOREIGN KEY REFERENCES Users(UserID), AccountType varchar(50), CreditBalance decimal);

CREATE TABLE Transactions (TransactionID int NOT NULL identity (1,1) PRIMARY KEY, UserAccountID int FOREIGN KEY REFERENCES UserAccount(UserAccountID), TransferAmount decimal, CreditOrWithdrawal bit NULL, TransferCode varchar(50), TransactionDateTime datetime);
```

### Populate Tables

```
USE UserCreditDB
GO

INSERT INTO [dbo].[Users] VALUES (1, 'Eugene', '22 Wendlin Ln'), (2, 'Bob Dole', '15 Kingsley Cir'), (3, 'Bob Barker', '34 Burbank Dr')

GO
```

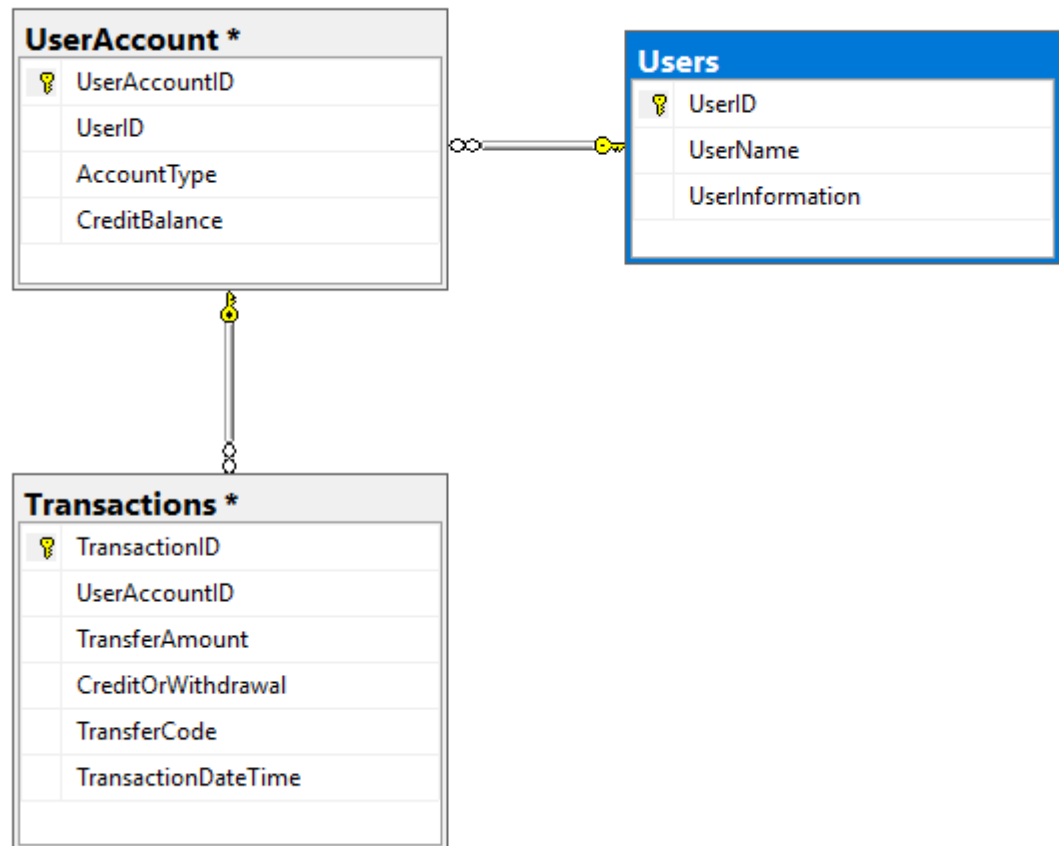
```

INSERT INTO [dbo].[UserAccount] VALUES(1, 1, 1, 5000), (2, 2, 1, 10000), (3, 3, 0, -
2000);

GO

```

## Database Diagram



## Create Stored Procedure for Log Transaction Attempt

```

USE UserCreditDB
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[LogTransactionAttempt]
(
    @AccountID INT,
    @Amount decimal,
    @CreditOrWithdrawal bit

```

```

)
AS
BEGIN
    INSERT INTO Transactions
    (
        UserAccountId,
        TransferAmount,
        CreditOrWithdrawal,
        TransactionDateTime
    )
    VALUES
    (
        @AccountID,
        @Amount,
        @CreditOrWithdrawal,
        GETDATE()
    )
    RETURN @@IDENTITY
END

```

## Create Stored Procedure for Perform Transaction

```

USE [UserCreditDB]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[performTransaction]
(
    @FromAccountId INT,
    @ToAccountId INT,
    @Amount FLOAT
)
AS
BEGIN
    DECLARE @FromAccountTest INT
    SELECT @FromAccountTest = (SELECT Count(*)
        FROM UserAccount
        WHERE UserAccountId = @FromAccountId)
    IF @FromAccountTest <> 1
    BEGIN
        RAISERROR('From Account Does Not Exist Error', 10, 1)
        RETURN
    END
    DECLARE @ToAccountTest INT
    SELECT @ToAccountTest = (SELECT Count(*)
        FROM UserAccount
        WHERE UserAccountId = @ToAccountId)
    IF @FromAccountTest <> 1
    BEGIN
        RAISERROR('To Account Does Not Exist Error', 10, 2)
    END

```

```

        RETURN
    END

    DECLARE @FromAccountBalance  FLOAT
    SELECT @FromAccountBalance = (SELECT CreditBalance
        FROM UserAccount
        WHERE UserAccountId = @FromAccountID)

    IF (@FromAccountBalance < @Amount)
    BEGIN
        RAISERROR('Insufficient Balance Error', 10, 3)
        RETURN
    END

    BEGIN TRANSACTION

    DECLARE @TransactionID INT

    EXEC @TransactionID = LogTransactionAttempt @FromAccountID, @Amount, 0
    IF (@TransactionID = 0)
    BEGIN
        ROLLBACK TRANSACTION
        RAISERROR('Withdrawal Error', 10, 3)
        RETURN
    END

    EXEC @TransactionID = LogTransactionAttempt @ToAccountID, @Amount, 1
    IF (@TransactionID = 0)
    BEGIN
        ROLLBACK TRANSACTION
        RAISERROR('Deposit Error', 10, 3)
        RETURN
    END

    UPDATE UserAccount
    SET CreditBalance = CreditBalance - @Amount
    WHERE UserAccountId = @FromAccountID

    IF (@@ERROR <> 0)
    BEGIN
        ROLLBACK TRANSACTION
        RAISERROR ('Subtraction Error', 10, 3)
        RETURN
    END

    UPDATE UserAccount
    SET CreditBalance = CreditBalance + @Amount
    WHERE UserAccountId = @ToAccountID
    IF (@@ERROR <> 0)
    BEGIN
        RAISERROR ('Addition Error', 10, 3)
        ROLLBACK TRANSACTION
        RETURN
    END

    COMMIT TRANSACTION
END

```

## Execute Stored Procedures

Execute Procedure - [dbo].[performTransaction]

Select a page

General


Script ? Help

Parameter	Data Type	Output Parameter	Pass Null Value	Value
@FromAccoun...	int	No	<input type="checkbox"/>	1
@ToAccountId	int	No	<input type="checkbox"/>	2
@Amount	float	No	<input type="checkbox"/>	200


Connection

Server:  
LAPTOP-N4N80KJ6

Connection:  
LAPTOP-N4N80KJ6\Yevge

 [View connection properties](#)

Progress

 Ready

OK Cancel

Execute Procedure - [dbo].[performTransaction]

Select a page

General

Script


Help

Parameter	Data Type	Output Parameter	Pass Null Value	Value
@FromAccoun...	int	No	<input type="checkbox"/>	2
@ToAccountId	int	No	<input type="checkbox"/>	1
@Amount	float	No	<input type="checkbox"/>	-500


Connection

Server:  
LAPTOP-N4N80KJ6

Connection:  
LAPTOP-N4N80KJ6\Yevge

 [View connection properties](#)

Progress

 Ready

OK

Cancel



Execute Procedure - [dbo].[performTransaction]

Select a page

General


Script Help

Parameter	Data Type	Output Parameter	Pass Null Value	Value
@FromAccoun...	int	No	<input type="checkbox"/>	2
@ToAccountId	int	No	<input type="checkbox"/>	3
@Amount	float	No	<input type="checkbox"/>	300


Connection

Server:  
LAPTOP-N4N80KJ6

Connection:  
LAPTOP-N4N80KJ6\Yevge

 [View connection properties](#)

Progress

 Ready

OK Cancel

Results

/***** Script for SelectTopNRows command from SSMS *****/						
<pre> SELECT TOP (1000) [TransactionID] , [UserAccountID] , [TransferAmount] , [CreditOrWithdrawal] , [TransferCode] , [TransactionDateTime] FROM [UserCreditDB].[dbo].[Transactions] </pre>						
100 %						
Results Messages						
	TransactionID	UserAccountID	TransferAmount	CreditOrWithdrawal	TransferCode	TransactionDateTime
1	1	1	200	0	NULL	2019-02-17 17:35:59.933
2	2	2	200	1	NULL	2019-02-17 17:35:59.933
3	3	2	-500	0	NULL	2019-02-17 17:37:19.680
4	4	1	-500	1	NULL	2019-02-17 17:37:19.680
5	5	2	300	0	NULL	2019-02-17 17:38:28.023
6	6	3	300	1	NULL	2019-02-17 17:38:28.023

## Insufficient Balance Error Example

```
USE [UserCreditDB]
GO

DECLARE @return_value int

EXEC @return_value = [dbo].[performTransaction]
    @FromAccountId = 3,
    @ToAccountId = 2,
    @Amount = 500

SELECT 'Return Value' = @return_value

GO
```

100 %

Results Messages

Insufficient Balance Error

(1 row affected)

### From Account Does Not Exist Error Example

```
USE [UserCreditDB]
GO

DECLARE @return_value int

EXEC @return_value = [dbo].[performTransaction]
    @FromAccountId = 4,
    @ToAccountId = 1,
    @Amount = 500

SELECT 'Return Value' = @return_value

GO
```

0 %

Results Messages

From Account Does Not Exist Error

(1 row affected)