

Course Database Implementation

Transactions

Order with BOL.COM

bol.com

Kies een categorie

Gratis verzending vanaf 20 euro, gratis retourneren en 30 dagen bedenktijd*

Musical & concert cadeau >

Winkelwagentje *Bestel nu en je bestelling wordt gratis verzonden!*

| Artikel | Levering | Prijs | Aantal | Totaal |
|--|----------------|---------|--------------------------------|--|
| Microsoft SQL Server 2012 T-SQL Fundamentals Itzik Ben-Gan Engels - Paperback | 3-4 werkdagen | € 31,99 | <input type="text" value="1"/> | € 31,99 <input type="button" value="Verwijder"/> |
| Microsoft SQL Server 2012 High-Performance T-SQL Using Window Functions Itzik Ben-Gan Engels - Paperback | 5-10 werkdagen | € 36,27 | | <input type="button" value="Verwijder"/> |
| Microsoft SQL Server 2012 Bible Aaron Nelson Engels - Paperback | 4-8 werkdagen | € 44,80 | | <input type="button" value="Verwijder"/> |

Verkoop: Books2Anywhere

Cadeaukaart- of kortingscode invoeren

Totaal artikel
Indicatie verz
Totaal

bol.com

Bestellen in drie eenvoudige stappen!

Winkelwagentje
3 artikelen - Aanpassen

- Microsoft SQL Server 2012 High-Performance T-SQL Using Window Functions
- Microsoft SQL Server 2012 T-SQL Fundamentals
- Microsoft SQL Server 2012 Bible

Totaal artikelen 3 € 113,06
Verzendkosten 0 **Gratis**
Nog te betalen € 113,06

Mijn account Bestelstatus Klantenservice

1. Mijn gegevens 2. Betalen 3. Bevestiging

Waar wil je de bestelling laten bezorgen?

HAN
Dhr. M Nabben
Ruitenberglaan 26
6826 CC ARNHEM

[Andere bezorgadres kiezen](#)

Klaar om te bestellen?

bol.com

Bestellen in drie eenvoudige stappen!

Verwacht bezorgmoment
De artikelen in deze bestelling kennen v
 • 3-4 werkdagen (1 artikel)
 • 4-8 werkdagen (1 artikel)
 • 5-10 werkdagen (1 artikel)

Bezorgadres - Wijzig
HAN
Dhr. M Nabben
Ruitenberglaan 26
6826 CC ARNHEM

Als je op 'Naar betalen' klikt, ga je akkoord met Voorwaarden kopen bij andere verkopers. De Books2Anywhere. E-mails tussen jou en de verkoper, onder andere om te helpen als het nog

Winkelwagentje

- Microsoft SQL Server 2012 High-Performance T-SQL Using Window Functions
- Microsoft SQL Server 2012 T-SQL Fundamentals
- Microsoft SQL Server 2012 Bible

Totaal artikelen 3 € 113,06
Verzendkosten 0 **Gratis**
Nog te betalen € 113,06

Mijn account Bestelstatus Klantenservice

1. Mijn gegevens 2. Betalen 3. Bevestiging

Kies een betaalmethode

Heb je een kortingscode of cadeaukaart?

☐ **Achteraf betalen** - Betaal na ontvangst van de artikel(en)

☒ **IDEAL** - Online betalen via je Nederlandse bank
Meest gebruikte betaalmethode in Nederland
 kies je bank

☐ **Creditcard** - Snel en eenvoudig betalen

Klik op Bestelling plaatsen om de bestelling definitief te maken.

Payment by IDEAL

bol.com*Bestellen in drie eenvoudige stappen!***Winkelwagentje**Microsoft SQL Server 2012
High-Performance T- ...Microsoft SQL Server 2012
T-SQL FundamentalsMicrosoft SQL Server 2012
BibleTotaal artikelen (3) € 113,06
Verzendkosten 0 **Gratis**

Nog te betalen € 113,06

Mijn account Bestelstatus Klantenservice

1. Mijn gegevens

2. Betalen

3. Bevestiging

Kies een betaalmethode**Heb je een kortingscode of cadeaukaart?**

Kortingscode of cadeaukaartcode

Toevoegen



> Air Miles inwisselen

Achteraf betalen - Betaal na ontvangst van de artikel(en)

iDEAL - Online betalen via je Nederlandse bank
Meest gebruikte betaalmethode in Nederland

kies je bank

Rabobank



Creditcard - Snel en eenvoudig betalen



Bestelling plaatsen

Klik op Bestelling plaatsen om de bestelling definitief te

Betalen met iDEAL

Ondertekenen Betalen Bevestigen Terug naar webwinkel

Begunstigde bol.com b.v. inzake bol.com

Omschrijving bol.com bestelling 8561872300

Bedrag € 113,06

Rekeningnummer

NL** RABO 0

Pasnummer

Betalen met



Random Reader



Rabo Scanner

Ga verder

Annuleren

Help

Ga alleen verder als de adresregel begint met http:

> Hoe controleert u de veiligheid van uw verbinding

> Lees meer over veiligheid

**Betalen met iDEAL**

Signeren Terug naar webwinkel

Er is een fout opgetreden

U heeft de betaling geannuleerd. U kunt terug gaan naar de webwinkel om de betaling alsnog te doen. (1050)

Klik op "Beëindigen" om terug te gaan naar de webwinkel.

Beëindigen

Confirmation of BOL.COM



Betalen met iDEAL



[Signeren](#) [Terug naar webwinkel](#)

Er is een fout opgetreden

U heeft de betaling geannuleerd. U kunt terug gaan naar de webwinkel om de betaling alsnog te doen. (1050)

Klik op "Beëindigen" om terug te gaan naar de webwinkel.

[Beëindigen](#)

bol.com

Bestellen in drie eenvoudige stappen!

Winkelwagentje



Microsoft SQL Server 2012
High-Performance T- ...



Microsoft SQL Server 2012
T-SQL Fundamentals



Microsoft SQL Server 2012
Bible

Totaal artikelen (3) € 113,06
Verzendkosten [?](#) **Gratis**

Nog te betalen € 113,06

[Mijn account](#) [Bestelstatus](#) [Klantenservice](#)

1. Mijn gegevens

2. Betalen

3. Bevestiging

! We vragen je aandacht voor het volgende:

- Je hebt de betaling met iDEAL afgebroken. Kies alsjeblieft een andere betaalmethode of probeer het opnieuw.

Kies een betaalmethode

Heb je een kortingscode of cadeaukaart?

[Toevoegen](#)



[Air Miles inwisselen](#)

☐ [Achteraf betalen](#) - Betaal na ontvangst van de artikel(en)

☐ **iDEAL** - Online betalen via je Nederlandse bank
Meest gebruikte betaalmethode in Nederland

☐ **Creditcard** - Snel en eenvoudig betalen



[Nieuwe kaart toevoegen](#)

Bestelling plaatsen

Klik op Bestelling plaatsen om de bestelling definitief te maken.

Order process 1-3

- **Add articles to shopping basket**

- BOL.COM >> ADDTOBASKET

- **Place order in basket**

- BOL.COM >> PLACEORDER

- **Pay order**

- BOL.COM >> PAYORDER

- IDEAL >> PAYORDER ...

- *RABOBANK >> PAYORDER*

- *RABOBANK >> CONFIRMPAYMENT*

- IDEAL >> CONFIRMPAYMENT

- BOL.COM >> CONFIRMPAYMENT

**CRUCIAL
TRANSACTION**

- **Receive confirmation order**

- BOL.COM >> SENDCONFIRMATION

- **Receive articles**

Order process 2-3

- Add articles to shopping basket

- BOL.COM >> ADDTOBASKET

- Place order in basket

- **BOL.COM >> PLACEORDER**

**NESTED
TRANSACTIONS**

- Pay order

- **BOL.COM >> PAYORDER**

- **IDEAL >> PAYORDER ...**

- *RABOBANK >> PAYORDER* << ERROR

- *RABOBANK >> CONFIRMPAYMENT*

- IDEAL >> CONFIRMPAYMENT

- BOL.COM >> CONFIRMPAYMENT

- Receive confirmation order

- BOL.COM >> SENDCONFIRMATION

- Receive articles

Order process 3-3

- Add articles to shopping basket

- BOL.COM >> ADDTOBASKET

- Place order in basket

- **BOL.COM >> PLACEORDER**

**NESTED
TRANSACTIONS**

- Pay order

- **BOL.COM >> PAYORDER**

- **IDEAL >> PAYORDER ...**

- *RABOBANK >> PAYORDER* << **ERROR**

- *RABOBANK >> CONFIRMPAYMENT*

- IDEAL >> CONFIRMPAYMENT

- BOL.COM >> CONFIRMPAYMENT

- Receive confirmation order

- BOL.COM >> SENDCONFIRMATION

- Receive articles

CANCEL

Example transactions

```
CREATE TABLE checkings (  
    account VARCHAR(10) CONSTRAINT pk_checkings PRIMARY KEY,  
    Balance INT NOT NULL  
)
```

```
CREATE TABLE savings (  
    account VARCHAR(10) CONSTRAINT pk_savings PRIMARY KEY,  
    Balance INT NOT NULL  
)
```

```
INSERT checkings (account, balance) VALUES ('Sally', 5000)
```

```
INSERT savings (account, balance) VALUES ('Sally', 2000)
```


Why do we need Transactions?

Bank transfer:

BEGIN TRANSACTION

UPDATE checkings

SET balance = balance - 1000

WHERE account = 'Sally'

UPDATE savings

SET balance = balance + 1000

WHERE account = 'Sally'

COMMIT TRANSACTION

These statements must *both* execute, or *none* of them

Transaction management **may** (!) help you

Try out

BEGIN TRANSACTION

```
UPDATE checkings  
SET balance = balance - 1000  
WHERE account = 'Sally'
```

```
SELECT ch.balance + sav.balance AS [Total Balance]  
FROM checkings ch INNER JOIN savings sav  
    ON ch.account = sav.account  
WHERE ch.account = 'Sally'
```

```
/*
```

```
now break the connection without committing  
and execute the SELECT statement again
```

```
*/
```

Transaction

Set of database operations

- to be completed at one time
- as though they were a single, **atomic operation**

A transaction must either be

- ***wholly committed***
 - making the changes durable
- or **rolled back**
 - any changes must be undone

Commands are database vendor specific!

Transactions in SQL Server

Transactions are managed at the connection level by the Recovery Manager

Three modes:

- **Autocommit**
 - Default
- **Explicit transaction management**
 - For executing two or more SQL-statements in a transaction
- **Implicit transaction management**
 - For ANSI-compatibility

Autocommit mode

Default mode in SQL Server

EVERY T-SQL-statement is committed or rolled back when it completes

INSERT, UPDATE, DELETE, CREATE, etc.

- **UPDATE titles**

- SET price = price * 1.05**

- **ALL rows will be committed or NO rows at all! (including execution of triggers)**
- **THROW in Trigger causes a Rollback (RAISERROR doesn't !!)**

Explicit Transactions

Programmer uses specific instructions:

- BEGIN TRANSACTION
- COMMIT TRANSACTION
- ROLLBACK TRANSACTION

- SAVE TRANSACTION *savepoint_name*
- ROLLBACK TRANSACTION *savepoint_name*

BEGIN TRANSACTION

INSERT *orders* ...

UPDATE *inventory*

SET ...

COMMIT TRANSACTION, or ROLLBACK TRANSACTION

'Nested' Transactions

Transactions can be 'nested'

- Usually occurs when stored procedures or triggers with **BEGIN TRAN – COMMIT TRAN** statements invoke one another
- **BEGIN TRAN**
 - ...
 - BEGIN TRAN**
 - ...
 - COMMIT TRAN**
 - ...
 - COMMIT TRAN**

Only the outermost BEGIN-COMMIT statement pair applies!!

But ROLLBACK TRAN rolls back ALL changes up to first BEGIN TRAN.

@@TRANCOUNT

Tells how many **BEGIN TRAN** commands have been executed without being committed

When no open transactions exist: **@@TRANCOUNT is 0**

BEGIN TRAN: **@@TRANCOUNT += 1**

COMMIT TRAN: **@@TRANCOUNT -= 1**

ROLLBACK TRAN: **@@TRANCOUNT := 0**

SAVE TRAN savepoint_name: *no effect on @@TRANCOUNT*

ROLLBACK TRAN savepoint_name: *no effect on @@TRANCOUNT*

Implicit Transactions

Command:

`SET IMPLICIT_TRANSACTIONS ON`

Result:

SQL Server then automatically starts a transaction when it first executes:

- `DELETE, INSERT, SELECT, UPDATE, CREATE, DROP, ALTER`, etc.
- transaction remains in effect until a `COMMIT` or `ROLLBACK` is issued

For compatibility with Oracle, DB2, ...

Error handling inside transactions

Transaction management **does not handle errors!**

– Example scenario:

BEGIN TRANSACTION

INSERT ...

INSERT ...

INSERT ...

COMMIT TRANSACTION

executed

error, not executed

Executed!!!

Violates the Atomic property

EXCEPTION HANDLING is REQUIRED !!!

Without exception handling

```
CREATE TABLE test (  
    col INT CONSTRAINT PK_TEST PRIMARY KEY  
)  
GO
```

```
BEGIN TRAN  
    INSERT test (col) VALUES (1)  
    INSERT test (col) VALUES (1)  
    INSERT test (col) VALUES (2)  
COMMIT TRAN
```

```
-- Messages? Exercise 1
```

Is this exception handling?

```
BEGIN TRAN
  DECLARE @err int
  INSERT test (col) VALUES (1)
  SET @err = @@ERROR
  IF @err <> 0 ROLLBACK TRAN

  INSERT test (col) VALUES (1)
  SET @err = @@ERROR
  IF @err <> 0 ROLLBACK TRAN

  INSERT test (col) VALUES (2)
  SET @err = @@ERROR
  IF @err <> 0 ROLLBACK TRAN
COMMIT TRAN
```

-- Messages? Exercise 2

Exception handling with TRY/CATCH

```
BEGIN TRY
    BEGIN TRAN
        INSERT test (col) VALUES (1)
        INSERT test (col) VALUES (1)
        INSERT test (col) VALUES (2)
    COMMIT TRAN
END TRY
BEGIN CATCH
    ROLLBACK TRAN
END CATCH
```

-- Messages? Exercise 3

But when it's a nested transaction?

```
BEGIN TRAN
  BEGIN TRY
    BEGIN TRAN
      INSERT test (col) VALUES (1)
      INSERT test (col) VALUES (1)
      INSERT test (col) VALUES (2)
    COMMIT TRAN
  END TRY
  BEGIN CATCH
    ROLLBACK TRAN
  END CATCH
COMMIT TRAN
```

-- Messages? Exercise 4

Now as a stored proc...

```
CREATE PROC tranTest  
AS  
BEGIN
```

```
    BEGIN TRY  
        BEGIN TRAN  
            INSERT test (col) VALUES (1)  
            INSERT test (col) VALUES (1)  
            INSERT test (col) VALUES (2)  
            COMMIT TRAN
```

```
    END TRY  
    BEGIN CATCH  
        ROLLBACK TRAN  
    END CATCH
```

```
END  
GO
```

```
BEGIN TRAN -- suppose the call is inside a transaction:  
    EXEC tranTest  
COMMIT TRAN -- Messages? Exercise 5
```

Solution? **Exercise 6**

Make a generally usable template for transaction management in a stored procedure.

Explanation:

Stored procedures may need to use explicit transaction management.

Of course **TRY/CATCH** is used for exception handling.

But the sp itself may be called from another transaction, what can be checked in the sp.

In this case it is preferred that the sp itself does not execute a **BEGIN TRAN**, but instead a **SAVE TRAN**: the sp can do a rollback to the savepoint, but leave the actual rollback or commit to the *calling transaction*.

Exception handling in transactions

Guidelines for stored procedures

Commit as many transactions as you begin

Don't rollback a transaction unless

@@TRANCOUNT was zero when the sp started

- Better not to rollback at all
- Protects you from the situation where a sp is executed in another transaction
- The stored procedure must signal the **caller** that it has failed

Transaction Log

```
CREATE DATABASE dbTest
```

Results in creation of **two** files:

| | |
|----------------|------------------------|
| dbTest.mdf | -- the database |
| dbTest_log.ldf | -- the transaction log |

Data changes do NOT go initially to the database itself

Changes are **first** written serially to the **Transaction Log**

Transaction Log

Write-ahead log

- on the database level
- so **ONE** log for all users of the database

The log is changed in cache, and is only *flushed* to disk at a **COMMIT**

SQL Server uses the log to recover data in case of

- a system failure
- a transaction cancellation request

Data Modification

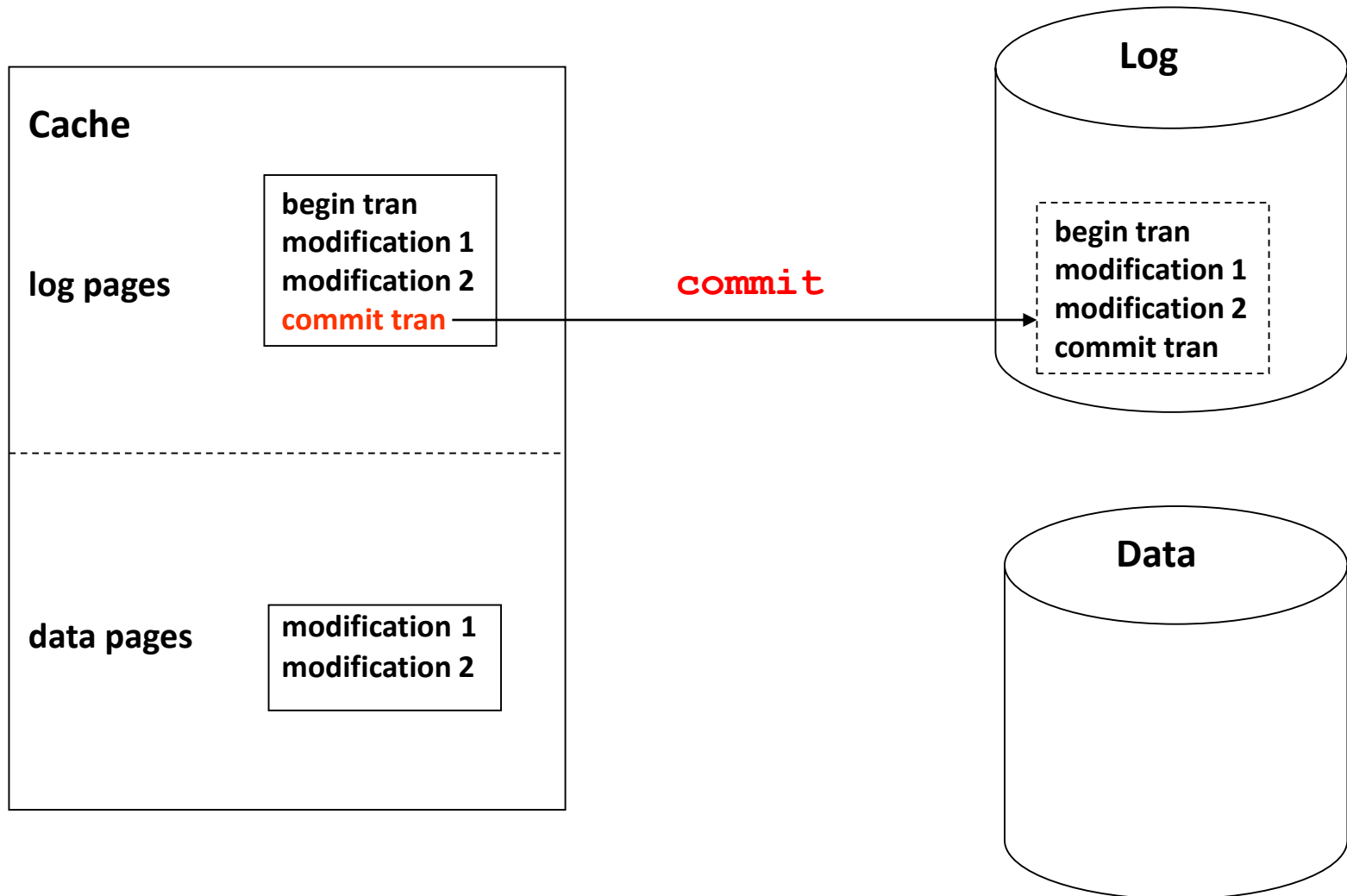
SQL Server caches modifications in buffers for a period of time to optimize disk writes:

- **BEGIN TRANSACTION** written in log (log cache)
- Modification in log (log cache)
- Modification of the data (data cache)
- **COMMIT TRANSACTION** in log (log cache)

The **COMMIT** forces a 'Flush' of dirty log pages to disk

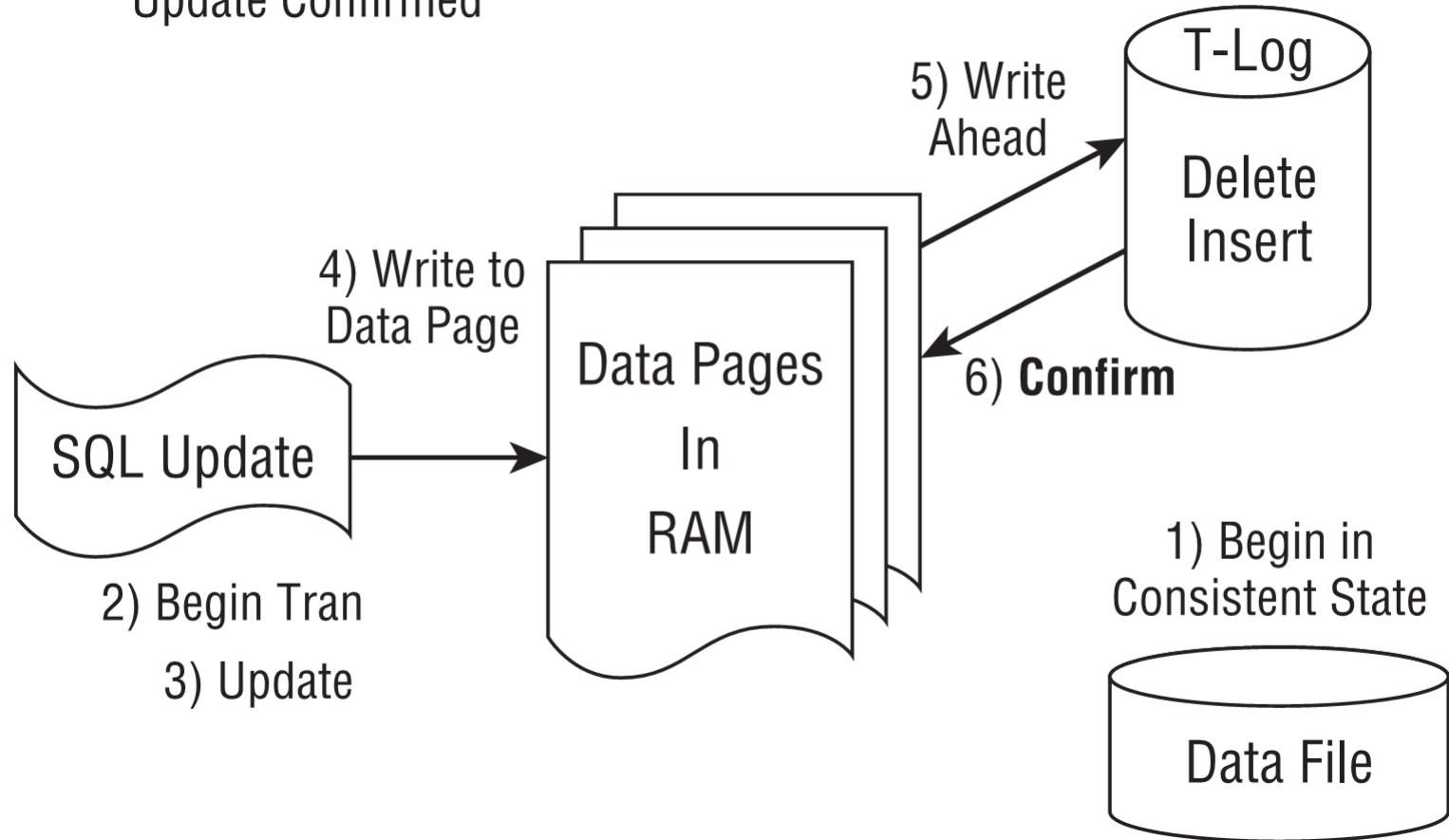
- *Dirty pages*: Log pages or data pages that have been modified, but are not yet been written to disk
- Log pages are written to disk *before* data pages are written to disk

The Commit Process

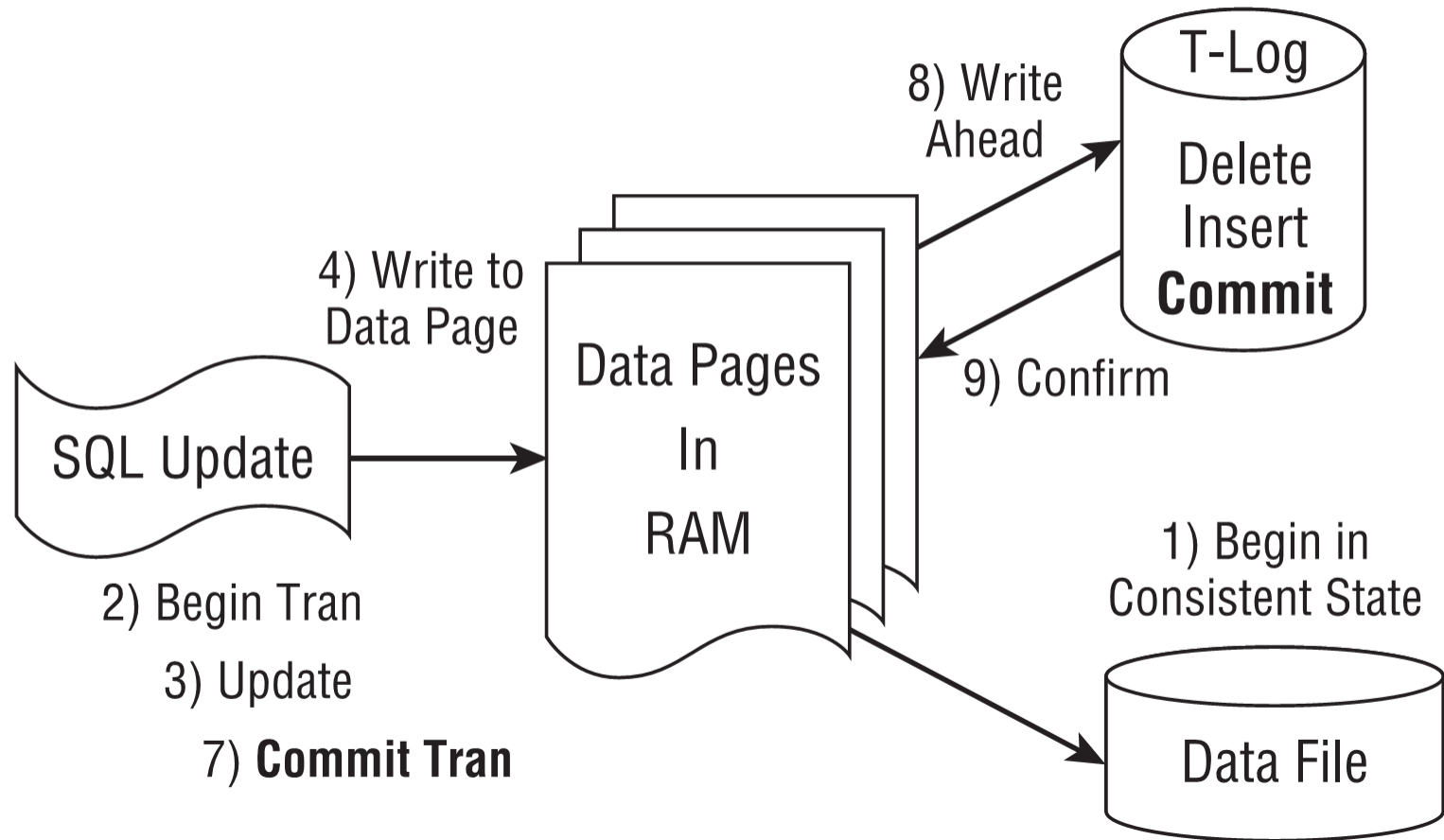


T Update

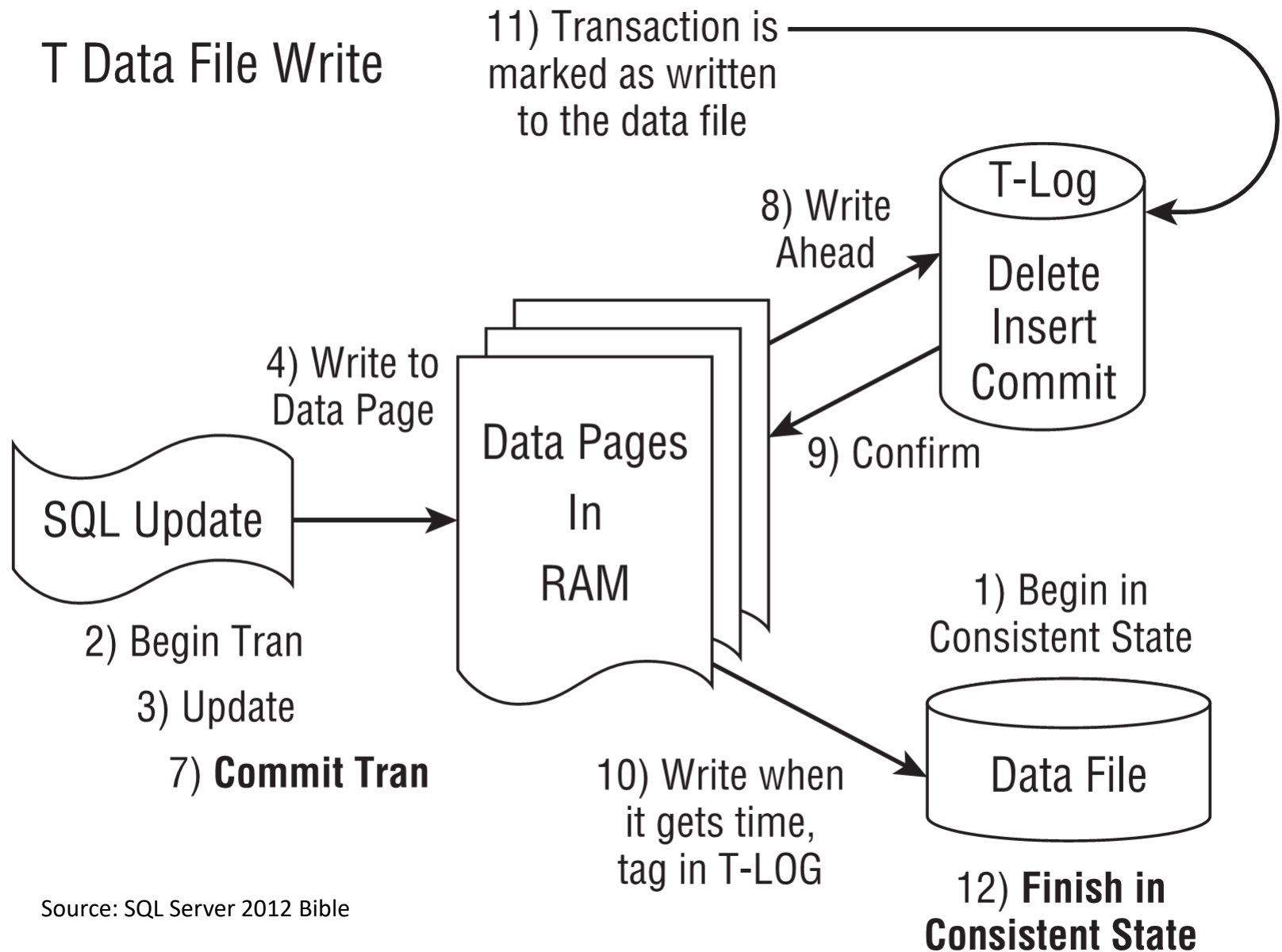
Update Confirmed



T Commit



T Data File Write



Checkpoint

periodic synchronization between database and log

- **ALL** dirty data pages are written from cache to disk
- log pages are written at COMMIT or ROLLBACK
- checkpoint is also being logged

reduces the time and resources needed to recover

- otherwise the log would fill up

checkpoint is issued by:

- DBO with the command **CHECKPOINT**
- Server when the *recovery interval* is elapsed

Recovery

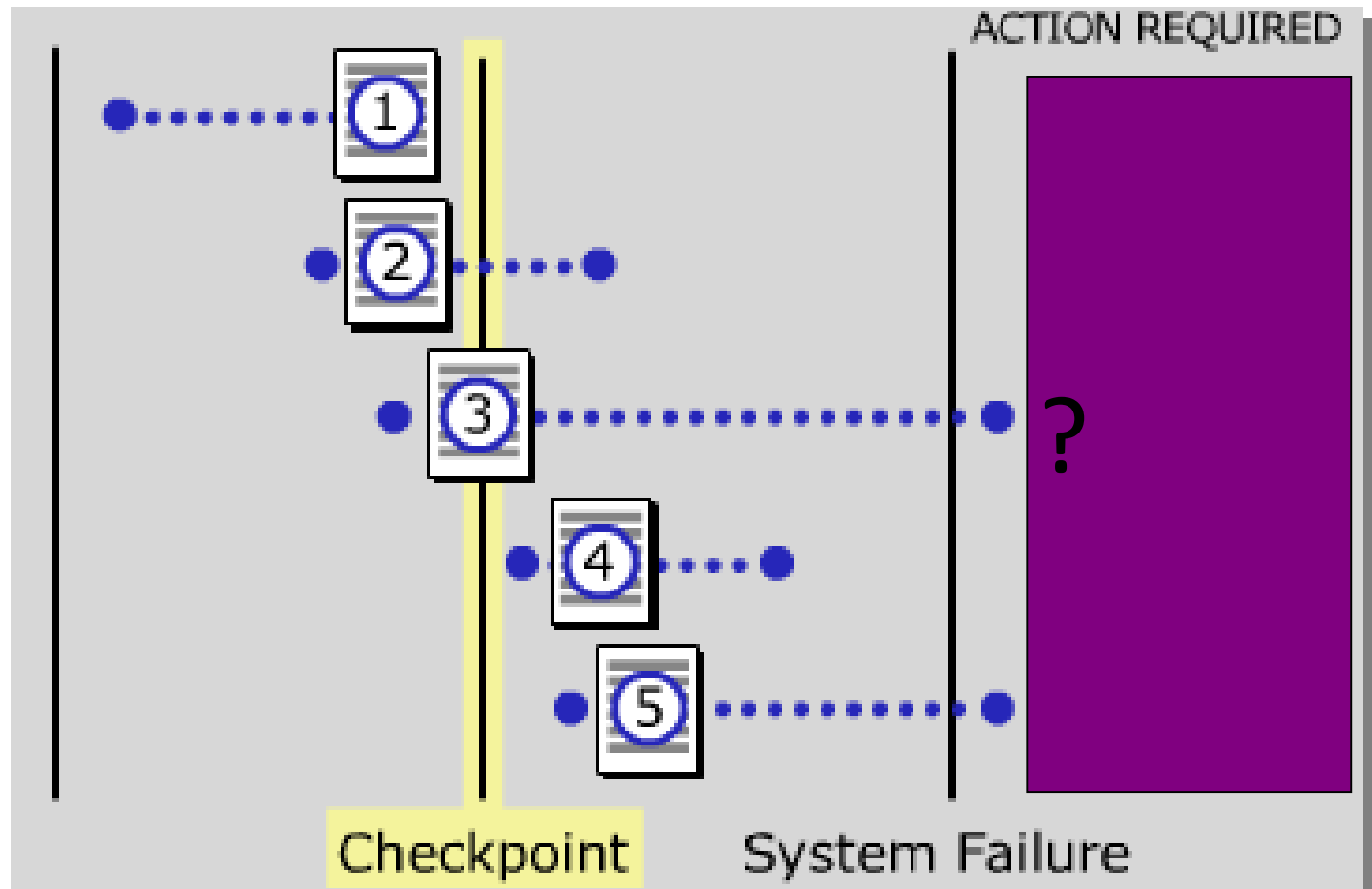
At recovery time the server examines the log on disk

Committed transactions which are not yet written to disk are written to disk (**roll forward**)

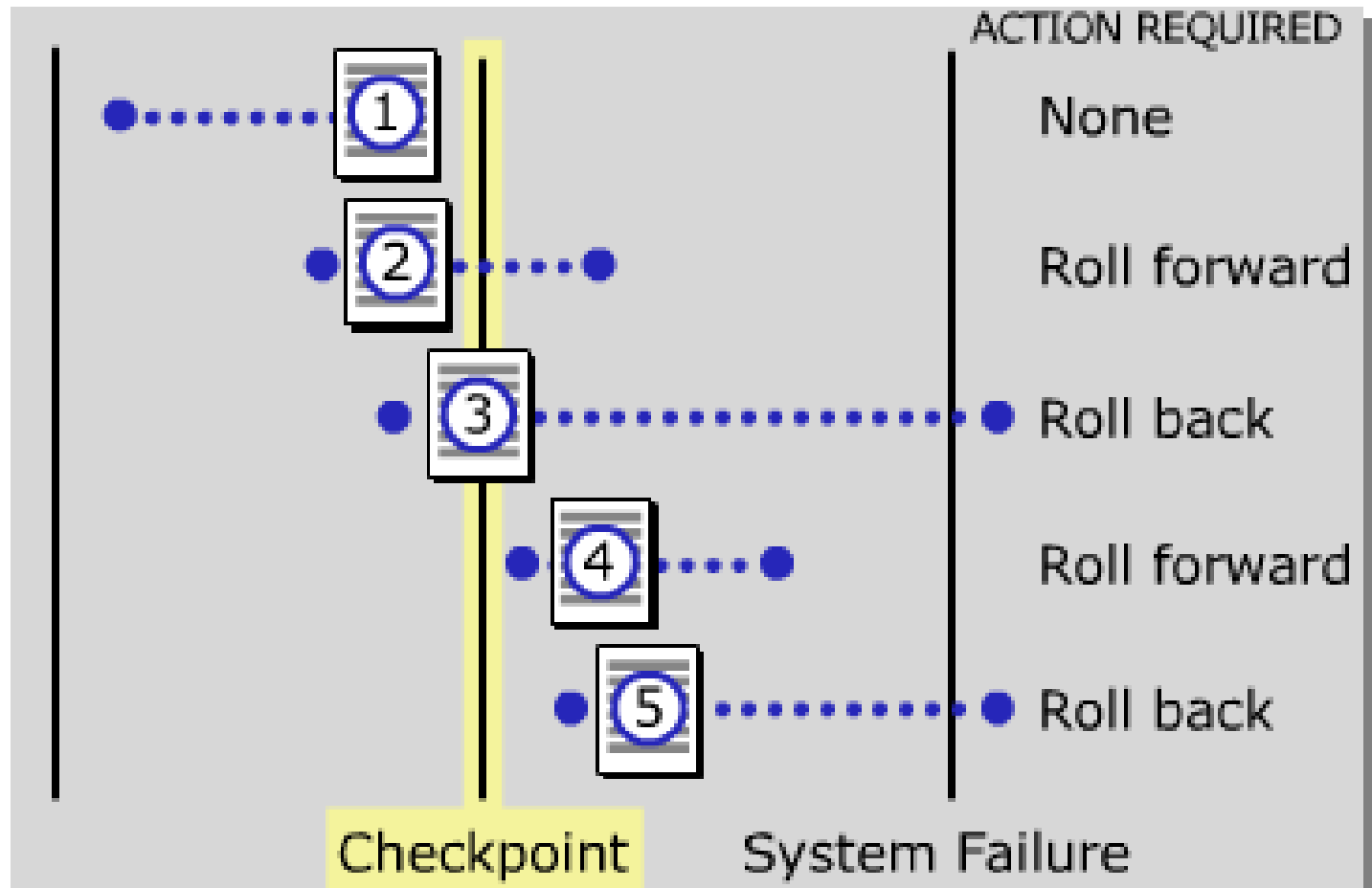
- Server looks for *checkpoints* recorded in the log

Uncommitted transactions are **rolled back**

Failure



Recovery



Recovery Actions

1. No action

- committed: log pages were flushed to log disk
- before the checkpoint: log and database were synchronized

2. Roll forward

- committed: log pages were flushed to log disk
- after the checkpoint: not all modifications are on data disk

3. Roll back

- not committed, and unknown where a commit was intended to be
- started before the checkpoint, so some modifications may be on data disk. some modifications after the checkpoint may be on log disk due to commits of other processes

4. Roll forward

- committed: log pages were flushed to log disk
- after the checkpoint: none of the modifications are on data disk

5. Roll back

- not committed, and unknown where a commit was intended to be
- started after the checkpoint, so some modifications may be on log disk due to commits of other processes

ACID Properties of Transactions

Atomicity

- A transaction is an atomic unit of processing, **all or nothing**
- Responsibility of the recovery manager to ensure completion
- Recovery happens every time at start-up

Consistency

- The transaction must take the database from one consistent state to another
- Responsibility of the database programmers

ACID Properties of Transactions

Isolation

- A transaction should **not** make its updates **visible** to other transactions **until it is committed**
- Enforced by the concurrency control method imposed by the programmers

Durability

- Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure
- Responsibility of the recovery manager