



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022

Transaction Management

Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales

www.uwtsd.ac.uk

So far...

1. Query Processing
2. Joins

Learning Objectives

1. Transactions & its properties
2. Commit, Rollback and savepoint

Database Transactions



Transactions are likely to contain many parts.

executing and managing transactions are important database system activities.

sales transaction parts?



All parts of a transaction must be successfully completed to prevent _____?_____ problems.

Transactions

- Defined as any **one execution of a user program** in a DBMS
- Transaction is a unit of data processing or a list of actions.
- E.g. At a bank → withdrawal or deposit of money, transfer of money from A's a/c to B's a/c.
- A Transaction would involve manipulation of one or more values in a database.



```
read(staffNo = x, salary)
salary = salary * 1.1
write(staffNo = x, salary)
```

Database **Read** and
write operation on X

Example Transaction

Transaction with
2 database operations (read and write)
1 non database operation
(salary = salary*1.1)

Delete the member of staff with a given staff number x

Inconsistent state

All updates not
completed

Referential integrity will be lost

What are the two possible
outcomes of this transaction?

```
delete(staffNo = x)
for all PropertyForRent records, pno
begin
    read(propertyNo = pno, staffNo)
    if (staffNo = x) then
        begin
            staffNo = newStaffNo
            write(propertyNo = pno, staffNo)
        end
    end
end
```

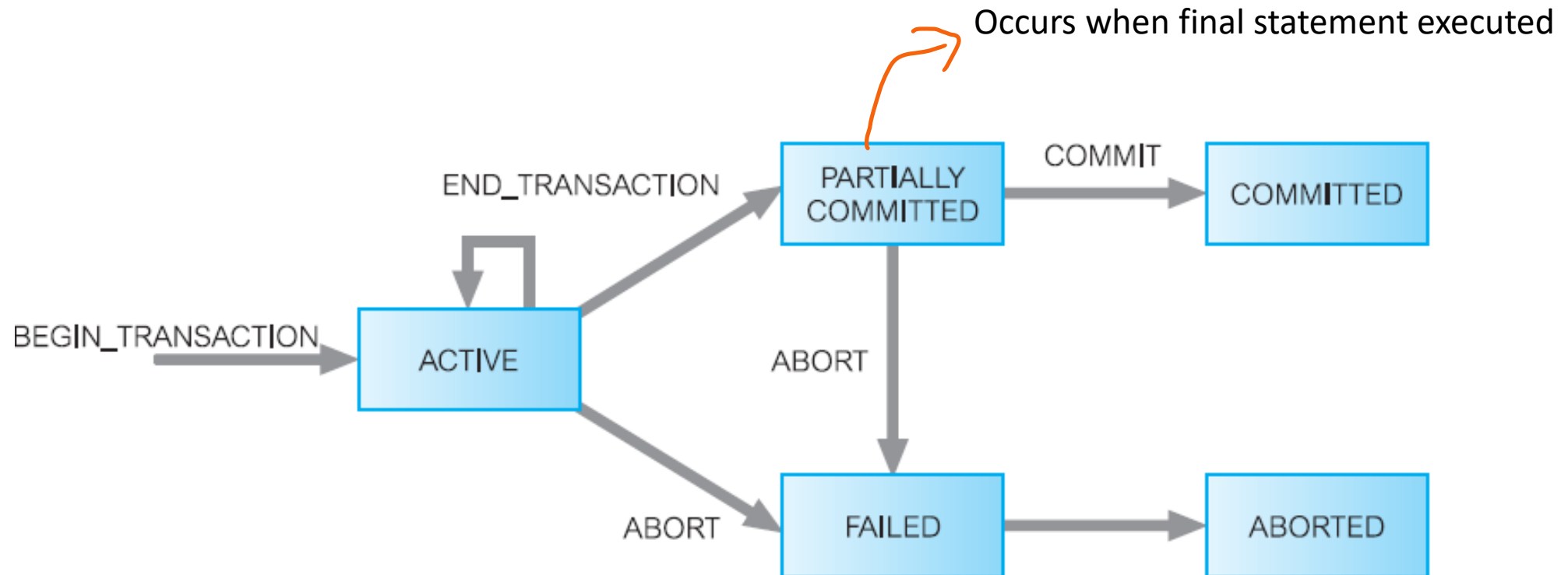
- Completes successfully- Transaction is said to have **committed**
Database reaches a new consistent state
- If the transaction does not execute successfully- **aborted**



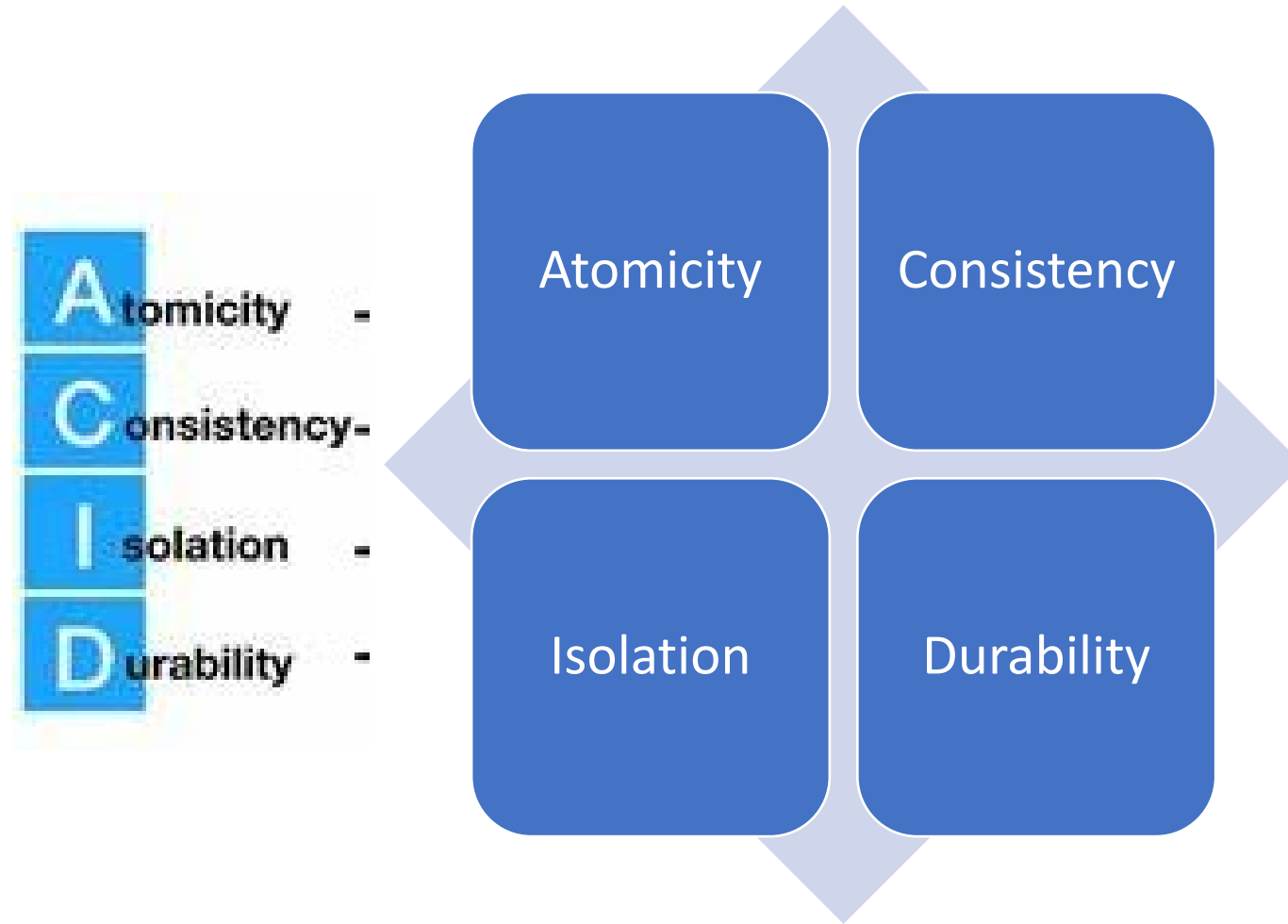
Rolled back or
undone(consistent state)

The keywords **BEGIN**

TRANSACTION, COMMIT, and ROLLBACK are available in many data manipulation languages to delimit transactions



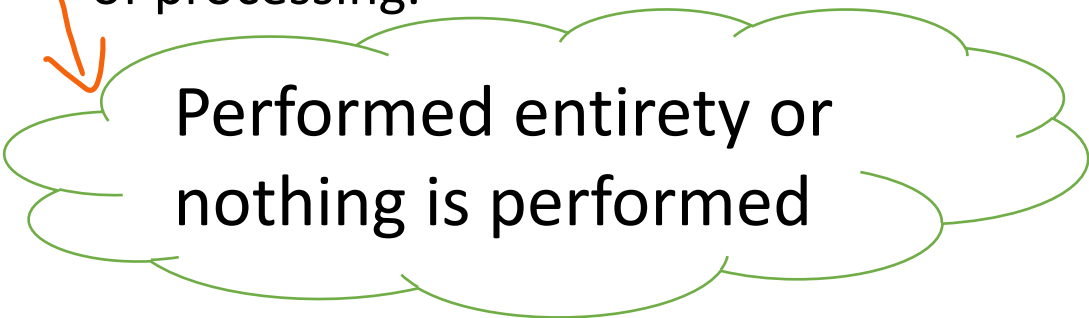
Properties of Transaction



Atomicity & Durability

•Atomicity

- Defines a transaction to be a single unit of processing.



Performed entirety or
nothing is performed

•Durability

- If a transaction is committed, its changes to the database are ***permanent***, even if there is a system failure.

The DBMS component that ensures atomicity and durability is called the **recovery manager**.

Consistency & Isolation

Consistency

- Ensures that when the transaction is completed, the DB **moves from one consistent state to another.**

Isolation

- Concurrent transactions are ***not aware*** of each other. Each thinks that it is the only running transaction.
- In other words, the updates of a transaction should not be visible till they are committed.

Database Consistency

Transfer £50 from account A to B :

read(A)

$A = A - 50$

write(A)

read(B)

$B = B + 50$

write(B)

Example: fund transfer

Transfer £50 from
account *A* to *B*:

read(*A*)

$A = A - 50$

write(*A*)

read(*B*)

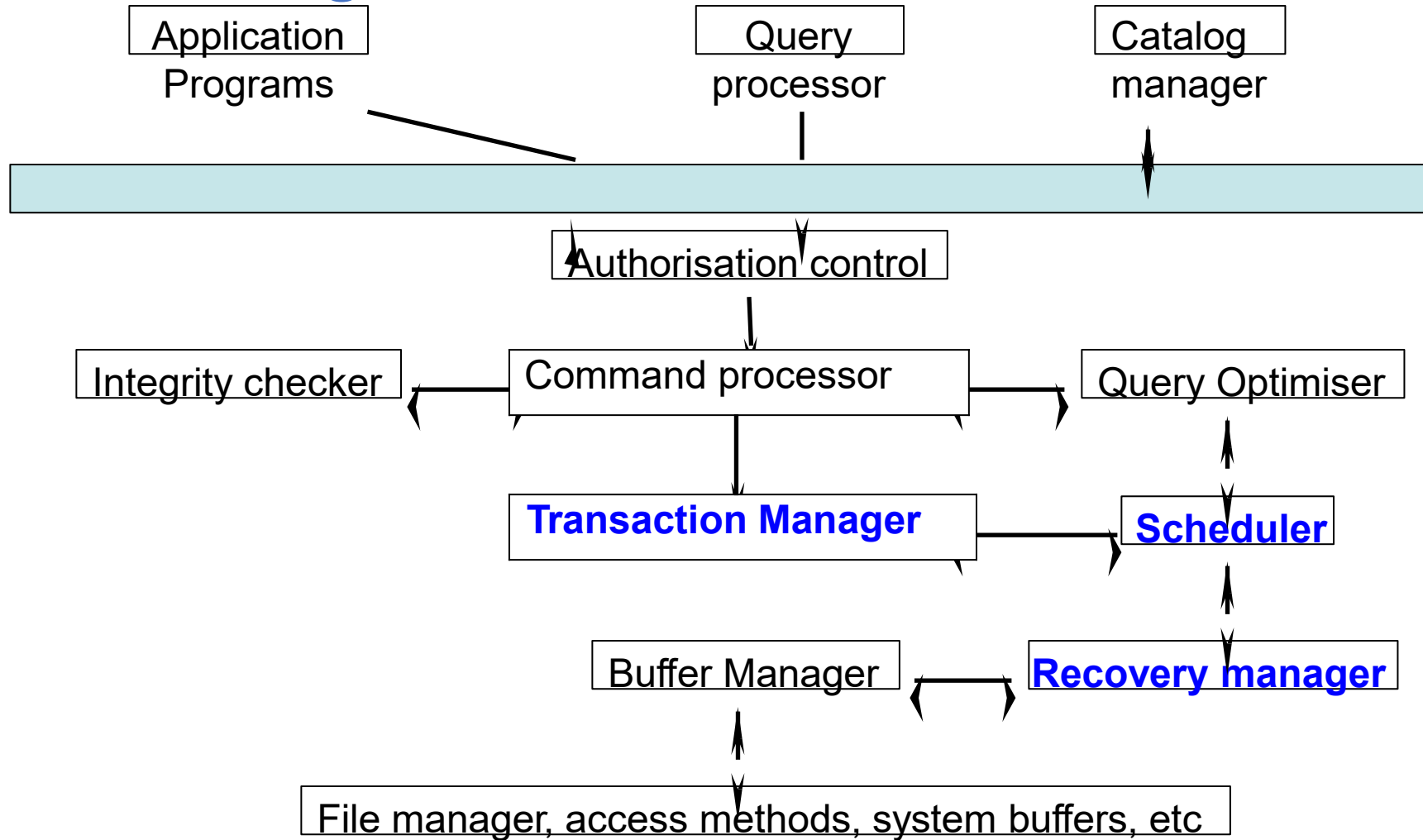
$B = B + 50$

write(*B*)

One transaction

- **Atomicity** – Shouldn't take money from *A* without giving it to *B*.
- **Consistency** – Money isn't lost or gained overall.
- **Isolation** – Other queries shouldn't see *A* or *B* change until completion.
- **Durability** – The money does not return to *A*, even after a system crash.

Database Manager



Transaction Manager

- The transaction manager enforces the ACID properties.
 - Schedules the operations of all transactions.
 - Uses COMMIT and ROLLBACK to ensure *atomicity*.
 - Locks or timestamps are used to ensure *consistency* and *isolation*.
 - A log is kept to ensure *durability*.

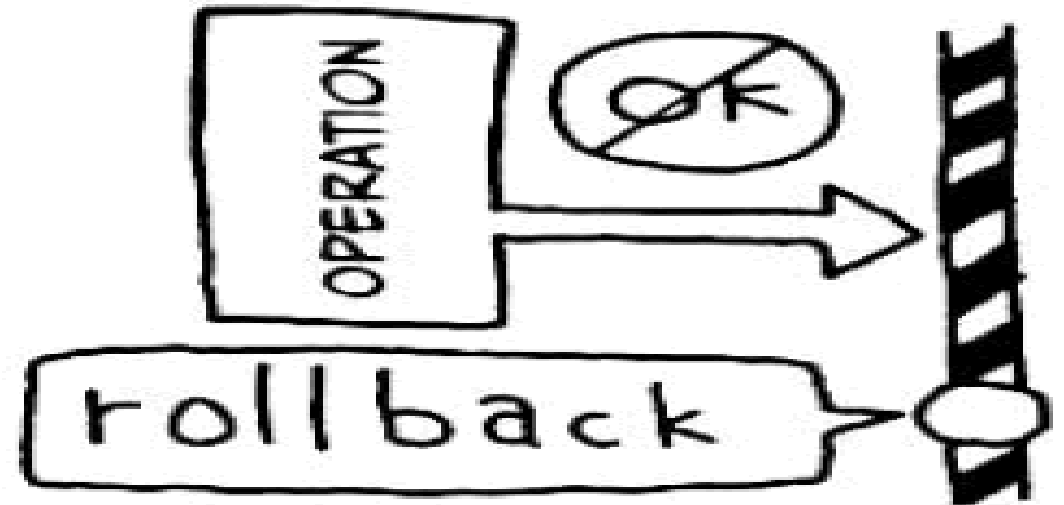
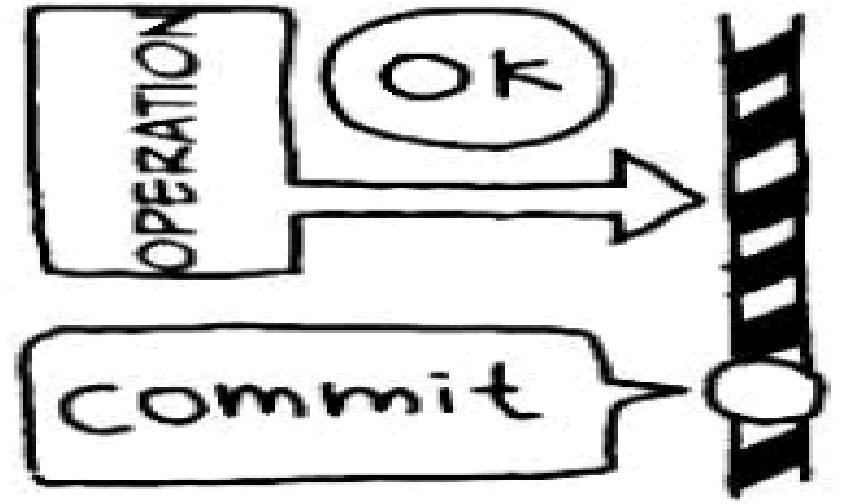
Atomicity

COMMIT - signal the *successful* end of a transaction

Any changes that have been made to the database should be made permanent. These changes are now available to other transactions.

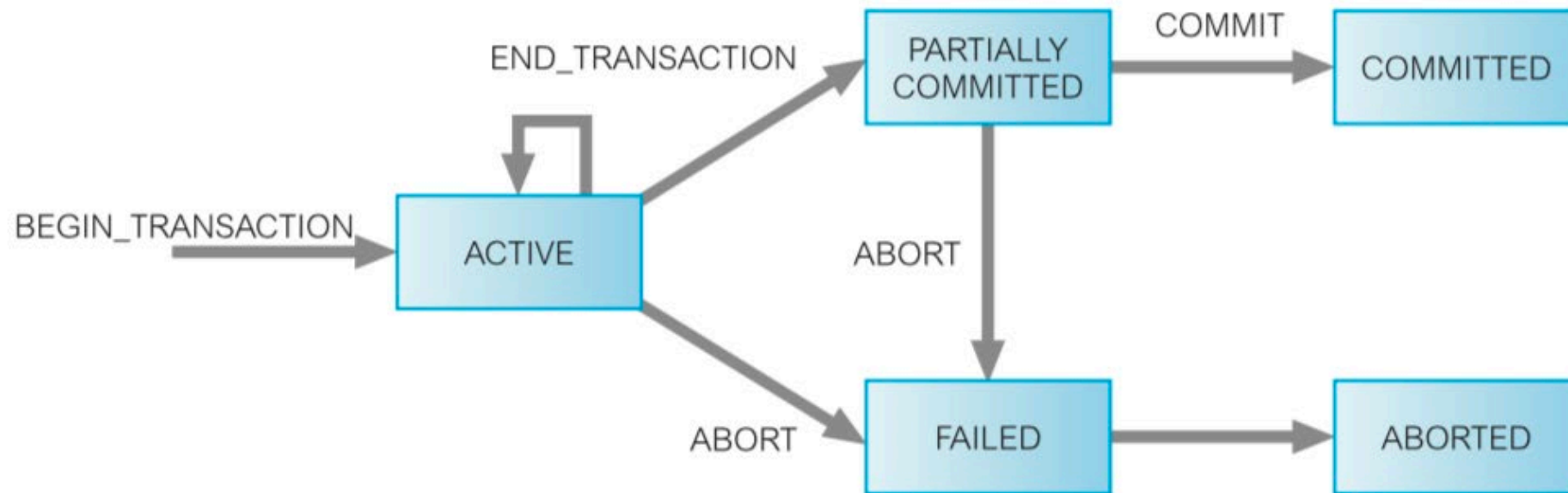
ROLLBACK - signal the *unsuccessful* end of a transaction

Any changes that have been made to the database should be undone. It is now as if the transaction never happened, it can now be reattempted if necessary.



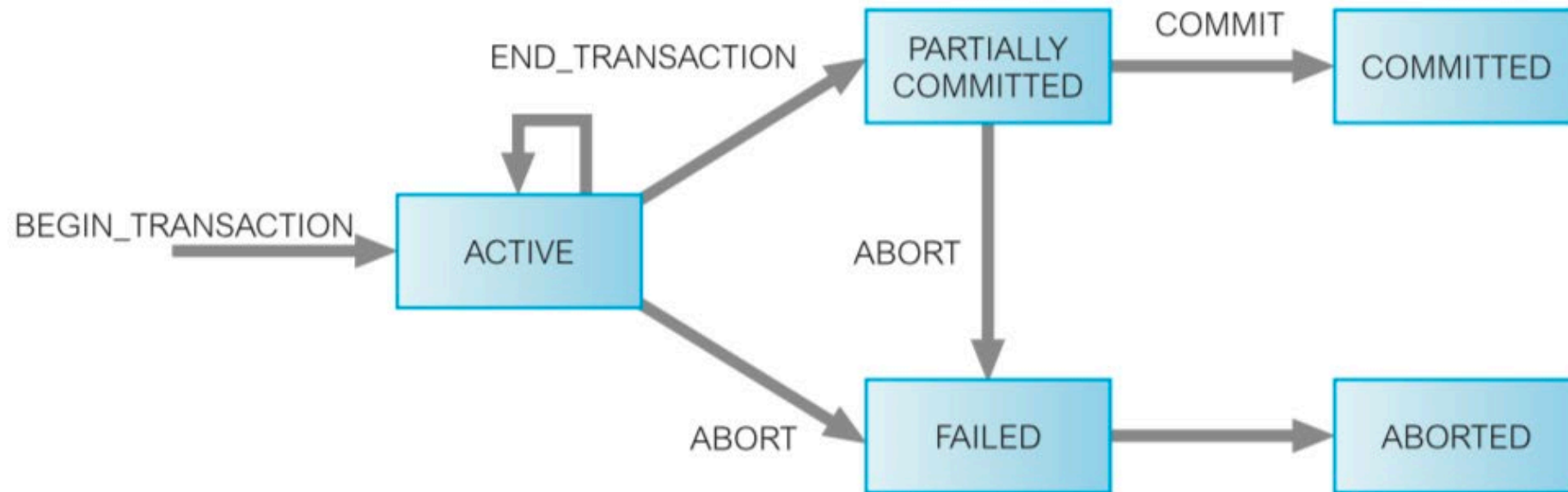
States of Execution

- A Transaction has many states of execution.
- State transition diagram for a transaction is given below.



States of Execution

- A Transaction has many states of execution.
- State transition diagram for a transaction is given below.



States of Execution - Example

Transaction T1

BEGIN TRANSACTION

READ(A);

A := A - 500;

WRITE(A);

READ(B);

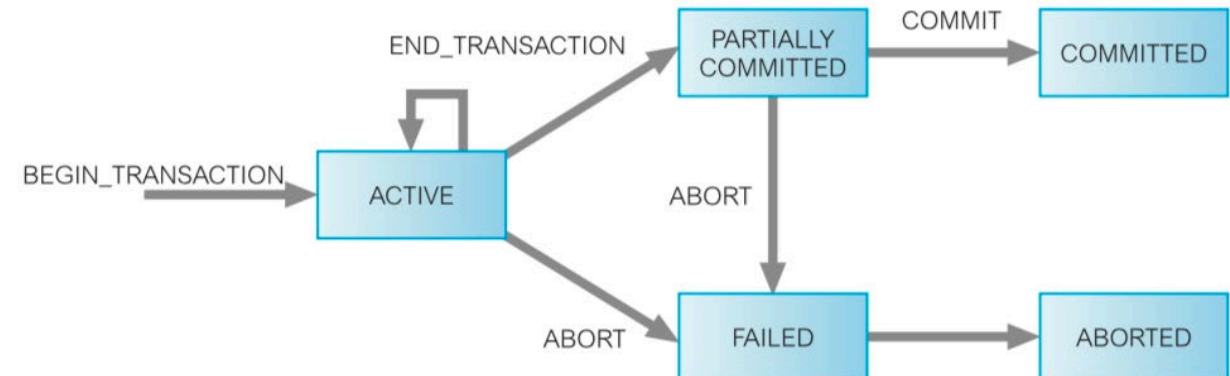
B := B + 500;

WRITE(B);

COMMIT;

END TRANSACTION;

State	A	B
ACTIVE	1000	1000
PARTIALLY COMMITTED	?	?



States of Execution - Example

Transaction T1

BEGIN TRANSACTION

READ(A);

A := A - 500;

WRITE(A);

READ(B);

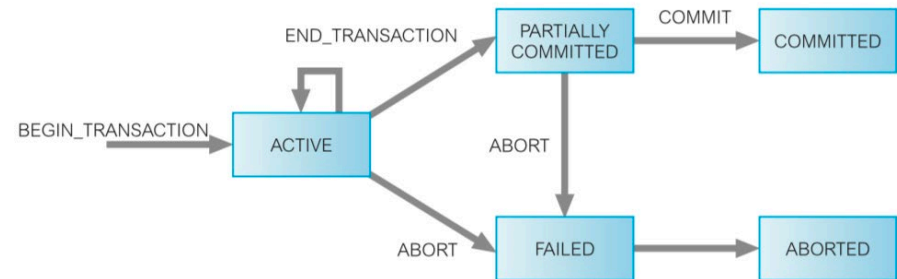
B := B + 500;

WRITE(B);

COMMIT;

END TRANSACTION;

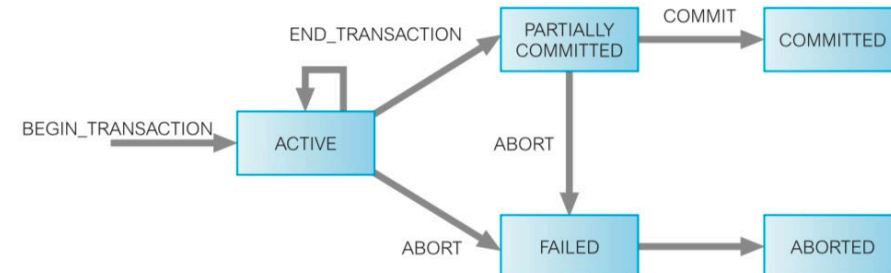
State	A	B
ACTIVE	1000	1000
PARTIALLY COMMITTED	500	1500



States of Execution - Example

Transaction T1
BEGIN TRANSACTION
READ(A);
A := A - 500;
WRITE(A);
READ(B);
B := B + 500;
WRITE(B);
COMMIT;
END TRANSACTION;

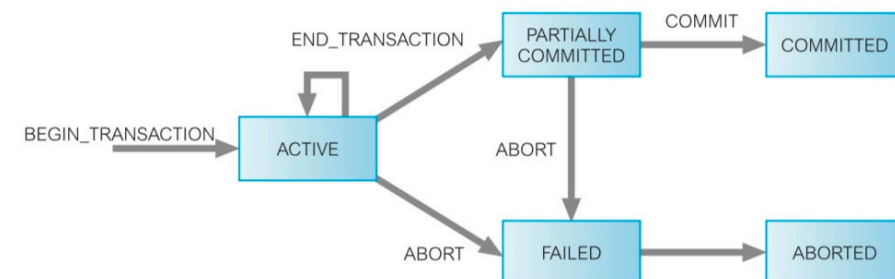
State	A	B
ACTIVE	1000	1000
PARTIALLY COMMITTED	500	1500
FAILED [before WRITE(A)]	?	?



States of Execution - Example

Transaction T1
BEGIN TRANSACTION
READ(A);
A := A - 500;
WRITE(A);
READ(B);
B := B + 500;
WRITE(B);
COMMIT;
END TRANSACTION;

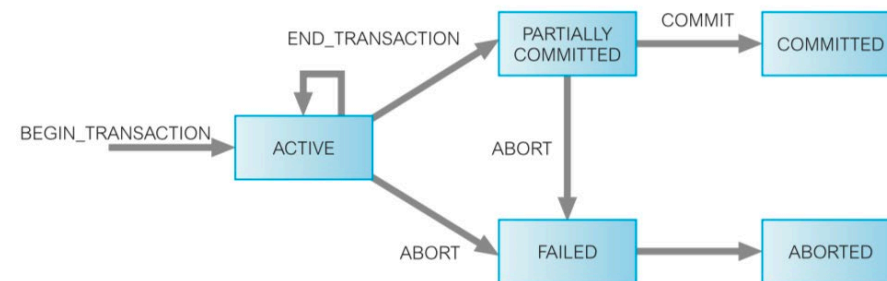
State	A	B
ACTIVE	1000	1000
PARTIALLY COMMITTED	500	1500
FAILED [before WRITE(A)]	1000	1000
FAILED [after WRITE(A)]	?	?



States of Execution - Example

Transaction T1
BEGIN TRANSACTION
READ(A);
A := A - 500;
WRITE(A);
READ(B);
B := B + 500;
WRITE(B);
COMMIT;
END TRANSACTION;

State	A	B
ACTIVE	1000	1000
PARTIALLY COMMITTED	500	1500
FAILED [before WRITE(A)]	1000	1000
FAILED [after WRITE(A)]	500	1000



States of Execution - Example

Transaction T1
BEGIN TRANSACTION
READ(A);
A := A - 500;
WRITE(A);
READ(B);
B := B + 500;
WRITE(B);
COMMIT;
END TRANSACTION;

State	A	B
ACTIVE	1000	1000
PARTIALLY COMMITTED	500	1500
FAILED [before WRITE(A)]	1000	1000
FAILED [after WRITE(A)]	500	1000
FAILED before COMMIT and after WRITE(B)	?	?

States of Execution - Example

Transaction T1
BEGIN TRANSACTION
READ(A);
A := A - 500;
WRITE(A);
READ(B);
B := B + 500;
WRITE(B);
COMMIT;
END TRANSACTION;

State	A	B
ACTIVE	1000	1000
PARTIALLY COMMITTED	500	1500
FAILED [before WRITE(A)]	1000	1000
FAILED [after WRITE(A)]	500	1000
FAILED before COMMIT and after WRITE(B)	500	1500
FAILED state from PARTIALLY COMMITTED state	?	?

States of Execution - Example

Transaction T1	State	A	B
BEGIN TRANSACTION			
READ(A);	ACTIVE	1000	1000
A := A - 500;	PARTIALLY COMMITTED	500	1500
WRITE(A);	FAILED [before WRITE(A)]	1000	1000
READ(B);			
B := B + 500;	FAILED [after WRITE(A)]	500	1000
WRITE(B);			
COMMIT;	FAILED before COMMIT and after WRITE(B)	500	1500
END TRANSACTION;	FAILED state from PARTIALLY COMMITTED state	500	1500

States of Execution - Example

Transaction T1
BEGIN TRANSACTION
READ(A);
A := A - 500;
WRITE(A);
READ(B);
B := B + 500;
WRITE(B);
COMMIT;
END TRANSACTION;

State	A	B
ACTIVE	1000	1000
PARTIALLY COMMITTED	500	1500
FAILED [before WRITE(A)]	1000	1000
FAILED [after WRITE(A)]	500	1000
FAILED before COMMIT and after WRITE(B)	500	1500
ABORTED state	?	?

States of Execution - Example

Transaction T1
BEGIN TRANSACTION
READ(A);
A := A - 500;
WRITE(A);
READ(B);
B := B + 500;
WRITE(B);
COMMIT;
END TRANSACTION;

State	A	B
ACTIVE	1000	1000
PARTIALLY COMMITTED	500	1500
FAILED [before WRITE(A)]	1000	1000
FAILED [after WRITE(A)]	500	1000
FAILED before COMMIT and after WRITE(B)	500	1500
ABORTED state	1000	1000

States of Execution - Example

Transaction T1	State	A	B
BEGIN TRANSACTION			
READ(A);	ACTIVE	1000	1000
A := A - 500;	PARTIALLY COMMITTED	500	1500
WRITE(A);	FAILED [before WRITE(A)]	1000	1000
READ(B);			
B := B + 500;	FAILED [after WRITE(A)]	500	1000
WRITE(B);			
COMMIT;	FAILED before COMMIT and after WRITE(B)	500	1500
END TRANSACTION;			
	COMMITTED state	?	?

States of Execution - Example

Transaction T1	State	A	B
BEGIN TRANSACTION			
READ(A);	ACTIVE	1000	1000
A := A - 500;	PARTIALLY COMMITTED	500	1500
WRITE(A);	FAILED [before WRITE(A)]	1000	1000
READ(B);			
B := B + 500;	FAILED [after WRITE(A)]	500	1000
WRITE(B);			
COMMIT;	FAILED before COMMIT and after WRITE(B)	500	1500
END TRANSACTION;	COMMITTED state	500	1500

Transaction States and DML (MySQL)

- The following are the MySQL's DML (Data Manipulation Language) related to the Transaction Management.
- **BEGIN TRANSACTION:** *indicates the start of a new transaction.*
- **COMMIT:** *commits the current transaction, making the changes permanent.*
- **ROLLBACK:** *rolls back the current transaction, cancelling the changes made.*
- **SET autocommit={0|1}:** *disables/enables the default autocommit mode for the current session. Default is set to 1 (hence, it is set to autocommit)*

Increasing salary by 10% in staff table

begin transaction;

Update staff

Set salary=salary*1.1

rollback transaction;

Activity

- Create a dummy database Book store with one table Books.
- The Books table has four columns: **id, name, category, and price.**

Datatypes:

Id: int

name: Varchar and not null

Category: Varchar and not null

Price: int not null

- Add 10 dummy records in books table for all fields

Executing multiple queries without using transactions

1. Insert new set of values into table

```
(15, 'Book15', 'Cat5', 2000)
```

2. Update books

3. Set price='25 hundred' where id=15

4. Delete from books

Where id=15

```
INSERT INTO Books
VALUES (15, 'Book15', 'Cat5', 2000)
UPDATE Books
SET price = '25 Hundred' WHERE id = 15
DELETE from Books
WHERE id = 15
```

Results		Messages		
	id	name	category	price
1	1	Book1	Cat1	1800
2	2	Book2	Cat2	1500
3	3	Book3	Cat3	2000
4	4	Book4	Cat4	1300
5	5	Book5	Cat5	1500
6	6	Book6	Cat6	5000
7	7	Book7	Cat7	8000
8	8	Book8	Cat8	5000
9	9	Book9	Cat9	5400
10	10	Book10	Cat10	3200
11	15	Book15	Cat5	2000

Automatically rollback SQL transactions

BEGIN TRANSACTION

INSERT INTO Books

VALUES (20, 'Book15', 'Cat5', 2000)

UPDATE Books

SET price = '25 Hundred' WHERE id = 20

DELETE from Books

WHERE id = 20

COMMIT TRANSACTION

Manually rollback SQL transactions

```
DECLARE @BookCount int
BEGIN TRANSACTION AddBook
    INSERT INTO Books VALUES (20, 'Book15', 'Cat5', 2000)
    SELECT @BookCount = COUNT(*) FROM Books WHERE name = 'Book15'
    IF @BookCount > 1
        BEGIN
            ROLLBACK TRANSACTION AddBook
            PRINT 'A book with the same name already exists'
        END
    ELSE
        BEGIN
            COMMIT TRANSACTION AddStudent
            PRINT 'New book added successfully'
        END
```

Variable declaration

SAVEPOINT Command

- SAVEPOINT creates points within the groups of transactions in which to ROLLBACK.

Syntax for Savepoint command:

SAVEPOINT SAVEPOINT_NAME;

- **Syntax for rolling back to Savepoint Command:**

ROLLBACK TO SAVEPOINT_NAME;

Question 1: Simple Commit

Suppose you have a table named Books with columns BookID, Title, Author, and Quantity. Perform the following operations:

Begin a transaction

Insert two new books into the Books table

Commit the transaction

Display all records in the Books table to verify the changes.

Question 2: Rollback Scenario

Consider the same Books table.

Start a transaction, update the quantity of a specific book to reflect new stock,

start a nested transaction, update the quantity again, and then rollback to the savepoint.

Finally, display the updated record for that book.

Question 3: Multiple Commits

Using the Books table, execute the following:

Begin a transaction

Insert a new book into the Books table

Commit the transaction

Start a new transaction

Update the author of a specific book

Commit this transaction

Display all records in the Books table to confirm changes.

Question 4: Rollback with Error Handling

Start a transaction in the Books table, delete a book that has a specific BookID, and then intentionally cause an error (e.g., trying to update a non-existing book). Catch the error and rollback the transaction, ensuring no changes are committed. Display all records in the Books table.

Question 5: Savepoint and Rollback

Begin a transaction in the Books table, insert multiple books, set a savepoint, update the quantity of a specific book, rollback to the savepoint, and commit the transaction. Display all records in the Books table to confirm the changes.

Summary

- Transactions
- Properties of Transactions
- States of execution of a transaction

Next Session...

Practice Queries

Assignment discussion