

# CSCI235 Database Systems

# Introduction to Transaction Processing (1)

Dr Janusz R. Getta

School of Computing and Information Technology -  
University of Wollongong

# Introduction to Transaction Processing

## Outline

[An interesting experiment](#)

[Where is a problem ?](#)

[Principles of transaction processing](#)

[Update synchronisation](#)

[ACID properties](#)

[Protocols](#)

# An interesting experiment

Use `psql` to create two simultaneous connections to the same user account

```
$psql -h localhost -U jrg -e
```

psql

```
$psql -h localhost -U jrg -e jrg
```

psql

Next, start the transactions in each connection and set **READ COMMITTED** isolation level for both transactions

```
jrg=> BEGIN;  
jrg=*> SET TRANSACTION ISOLATION LEVEL  
                READ COMMITTED;
```

Set isolation level and start the transactions

```
jrg=> BEGIN;  
jrg=*> SET TRANSACTION ISOLATION LEVEL  
                READ COMMITTED;
```

# An interesting experiment

Next, process the same `SELECT` statement in both connections

```
jrg=> SELECT COUNT(*) FROM DEPARTMENT;
```

```
count
```

```
-----  
                8  
(1 row)
```

Count the rows

```
jrg=> SELECT COUNT(*) FROM DEPARTMENT;
```

```
count
```

```
-----  
                8  
(1 row)
```

Count the rows

Obviously, the results are the same

# An interesting experiment

Now, we use one of the connections to **INSERT** a row into a relational table **DEPARTMENT**

```
jrg=> INSERT INTO DEPARTMENT VALUES('IT', 'SCIT', 5, 'James Krook', 1000);
```

Inserting a new department

And now repeat the same **SELECT** statements in both connections

```
jrg=> SELECT COUNT(*) FROM DEPARTMENT;
```

```
count
```

```
-----
          9
(1 row)
```

Count the rows

```
jrg=> SELECT COUNT(*) FROM DEPARTMENT;
```

```
count
```

```
-----
          8
(1 row)
```

Count the rows

Surprise, surprise, the results are different ! Why ?

In HTML view press 'p' to see the lecture notes

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Autumn 2024

5/21

# Introduction to Transaction Processing

## Outline

[An interesting experiment](#)

[Where is a problem ?](#)

[Principles of transaction processing](#)

[Update synchronisation](#)

[ACID properties](#)

[Protocols](#)

# Where is a problem ?

Why a modification performed by the first user is not visible to the second user ?

Is it correct that the second user must see all modifications performed by the first user ?

What if a modification performed by the first user is immediately visible to the second user and after that the first user rolls back the modification ?

Then, the second user is left with incorrect data !

Hence, only committed data can be revealed to the other users

Is such conclusion always true ?

## Problem statement

- Given a multiuser database system
- Find the most efficient synchronisation method for a set of concurrent processes accessing the shared database resources

# Introduction to Transaction Processing

## Outline

[An interesting experiment](#)

[Where is a problem ?](#)

[Principles of transaction processing](#)

[Update synchronisation](#)

[ACID properties](#)

[Protocols](#)



# Principles of transaction processing

A partially ordered set of **read, write** operations on the database items is called as a **transaction**

Users interact with a database by executing programs

Execution of a program is equivalent to execution of a partially ordered set of **read, write** operations

A database is visible to **transactions** as a collection of data items

Concurrently running **transactions** interleave their operations

**Transactions** have no impact on execution of their operations

Each **transaction** terminates by either **commit** or **abort** operation

Each **transaction** arrives at a consistent database state and must leave a database in a consistent state as well

# Principles of transaction processing

A sample concurrent processing of database transactions

Concurrent processing of database transactions		
T1	T2	
		x: \$100
a=read(x)		x: \$100 a: \$100
	b=read(x)	x: \$100 b: \$100
write(x, a-10)		x: \$90 a: \$100
	write(x, b+20)	x: \$120 b: \$100
	commit	x: \$120
commit		x: \$120

If a state of a bank account is \$100 then withdrawal of \$10 and deposit of \$20 cannot change a state of bank account to \$120

Uncontrolled concurrent processing of database transactions may **corrupt** a database

# Introduction to Transaction Processing

## Outline

[An interesting experiment](#)

[Where is a problem ?](#)

[Principles of transaction processing](#)

[Update synchronisation](#)

[ACID properties](#)

[Protocols](#)

# Update synchronisation

Database transaction can perform update in two different ways:

- A transaction immediately writes uncommitted values into a database - **update-in-place**
- A transaction does not modify a database until the time it commits itself - **deferred-update**

In the last example the transactions applied **update-in-place** to modify a database

A way how the transactions perform an update has no impact on the final outcomes, e.g. when deferred-update is applied a database still gets corrupted (see the next example)

# Principles of transaction processing

A sample concurrent processing of database transactions when **deferred-update** is applied

Concurrent processing of database transactions		
T1	T2	
		x: \$100
a=read(x)		x: \$100 a: \$100
	b=read(x)	x: \$100 b: \$100
write(x, a-10)		x: \$100 log T1:\$90
	write(x, b+20)	x: \$100 log T2:\$120
	commit	x: \$120
commit		x: \$90

If a state of a bank account is \$100 then withdrawal of \$10 and deposit of \$20 cannot change a state of bank account to \$90

**Deferred-update** does not solve the problem

# Introduction to Transaction Processing

## Outline

[An interesting experiment](#)

[Where is a problem ?](#)

[Principles of transaction processing](#)

[Update synchronisation](#)

[ACID properties](#)

[Protocols](#)

# ACID properties

Processing of database transactions must satisfy **ACID** properties

## Atomicity

- Each database operation is treated as a single unit (all-or-nothing)

## Consistency

- A transaction takes a database from one consistent state to another

## Isolation

- Transactions do not directly communicate one with each other and they do not read the intermediate results of the other transactions

## Durability

- The results of committed transactions must be permanent in a database in spite of failures

# Introduction to Transaction Processing

## Outline

[An interesting experiment](#)

[Where is a problem ?](#)

[Principles of transaction processing](#)

[Update synchronisation](#)

[ACID properties](#)

[Protocols](#)



# Protocols

An **execution atomicity protocol** ensures **Consistency** property

A **failure atomicity protocol** ensures **Atomicity, Isolation** and **Durability** properties

A sample incorrect **execution atomicity protocol**

Concurrent processing of database transactions		
T1	T2	
		x: \$100
a=read(x)		x: \$100 a: \$100
	b=read(x)	x: \$100 b: \$100
write(x,a-10)		x: \$90 a: \$100
	write(x,b+20)	x: \$120 b: \$100
	commit	x: \$120
commit		x: \$120

# Protocols

A sample incorrect **failure atomicity protocol**

Concurrent processing of database transactions		
T1	T2	
		x: \$100
a=read(x)		x: \$100 a: \$100
write(x,a-10)		x: \$90 a: \$100
	b=read(x)	x: \$90 b: \$90
	write(x,b+20)	x: \$110 b: \$90
	commit	x: \$110
abort		x: \$100

If a state of a bank account is \$100 then withdrawal of \$10 and deposit of \$20 cannot change a state of bank account to \$100

# Protocols

Execution atomicity protocol = Concurrency control protocol

Failure atomicity protocol = Recovery protocol

## Lost update problem

Concurrent processing of database transactions		
T1	T2	
		x: \$100
a=read(x)		x: \$100 a: \$100
	b=read(x)	x: \$100 b: \$100
write(x,a-10)		x: \$90 a: \$100
	write(x,b+20)	x: \$120 b: \$100
	commit	x: \$120
commit		x: \$120

In HTML view press 'p' to see the lecture notes

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Autumn 2024

19/21

# Protocols

## Inconsistent retrieval problem

		Concurrent processing of database transactions				
T1	T2	x	y	a	b	c
a=read(x)		100	50	100		
	b=read(y)	100	50	100	50	
write(x,a-10)		90	50	100	50	
	c=read(x)	90	50	100	50	90
write(y,a-30)		90	70	100	50	90
	print(b+c)	90	70	100	50	90

In HTML view press 'p' to see the lecture notes

[TOP](#)

Created by Janusz R. Getta, CSCI235 Database Systems, Autumn 2024

20/21

# References

T. Connolly, C. Begg, Database Systems, A Practical Approach to Design, Implementation, and Management, Chapter 22.1 Transaction Support, Chapter 22.2 Concurrency Control, Pearson Education Ltd, 2015