CS2102 Lecture 11
Transactions

Transactions: Recap

- Abstraction for representing a logical unit of work
- **ACID Properties**
    - Atomicity: Either all the effects of the transactions are reflected in the database or none are
    - Consistency: The execution of a transaction in isolation preserves the consistency of the database
    - Isolation: The execution of a transaction is isolated from the effects of other concurrent transaction executions
    - Durability: The effects of a committed transaction persists in the database even in the presence of system failures

Transaction Example: Money Transfer

```
int Transfer (int fromAcctId, int toAcctId, int amount)
{
    EXEC SQL BEGIN DECLARE SECTION;
        int fromBalance;    int toBalance;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL WHENEVER SQLERROR GOTO query_error;

    EXEC SQL SELECT balance INTO :fromBalance FROM Accounts
        WHERE accountId = :fromAcctId;
    if (fromBalance < amount) {
        EXEC SQL ROLLBACK;   return 1;
    }
    EXEC SQL SELECT balance INTO :toBalance FROM Accounts
        WHERE accountId = :toAcctId;
    EXEC SQL UPDATE Accounts SET balance = :toBalance + :amount
        WHERE accountId = :toAcctId;
    EXEC SQL UPDATE Accounts SET balance = :fromBalance - :amount
        WHERE accountId = :fromAcctId;
    EXEC SQL COMMIT;
    return 0;
    query_error: printf ("SQL error: %ld\n", sqlca->sqlcode); exit();
}
```

Transaction Example: Money Transfer

Two possible execution outcomes:

```
begin transaction;

select      balance into :fromBalance
from        Accounts
where       accountId = :fromAcctId;

rollback;
```

```
begin transaction;

select      balance into :fromBalance
from        Accounts
where       accountId = :fromAcctId;

select      balance into :toBalance
from        Accounts
where       accountId = :toAcctId;

update      Accounts
set         balance = :toBalance + :amount
where       accountId = 1;

update      Accounts
set         balance = :fromBalance - :amount
where       accountId = 2;

commit;
```

ANSI SQL Isolation Levels

- The isolation level for a transaction affects what the transaction will read
- ANSI SQL defines four isolation levels
  - Read Uncommitted (weakest isolation level)
  - Read Committed
  - Repeatable Read
  - Serializable (strongest isolation level)
- Choice of isolation level affects correctness vs performance tradeoff
- In many DBMSs, the default isolation level is Read Commited
- Configure using **set transaction isolation level** statement

Serial Transaction Executions

- Consider a set of transactions $S = \{T_1, \cdots, T_n\}$
- An execution of S is a serial execution if the execution of the transactions in S are not interleaved
  - For any pair of transactions T_i and T_j in S, either T_i completes execution before T_j starts its execution, or T_j completes execution before T_i starts its execution

Serial Transaction Executions

Two possible serial executions of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select      balance into :xbal
from        Accounts
where       accountId = 1;
select      balance into :ybal
from        Accounts
where       accountId = 2;
update      Accounts
set         balance = :ybal + 100
where       accountId = 2;
update      Accounts
set         balance = :xbal - 100
where       accountId = 1;
commit;
begin transaction;
select     balance into :ybal2
from       Accounts
where      accountId = 2;
select     balance into :xbal2
from       Accounts
where      accountId = 1;
update     Accounts
set        balance = :xbal2 + 100
where      accountId = 1;
update     Accounts
set        balance = :ybal2 - 100
where      accountId = 2;
commit;
```

```
begin transaction;
select     balance into :ybal2
from       Accounts
where      accountId = 2;
select     balance into :xbal2
from       Accounts
where      accountId = 1;
update     Accounts
set        balance = :xbal2 + 100
where      accountId = 1;
update     Accounts
set        balance = :ybal2 - 100
where      accountId = 2;
commit;
begin transaction;
select      balance into :xbal
from        Accounts
where       accountId = 1;
select      balance into :ybal
from        Accounts
where       accountId = 2;
update      Accounts
set         balance = :ybal + 100
where       accountId = 2;
update      Accounts
set         balance = :xbal - 100
where       accountId = 1;
commit;
```

Interleaved Transaction Executions

An interleaved execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select    balance into :xbal
from      Accounts
where     accountId = 1;
select    balance into :ybal
from      Accounts
where     accountId = 2;
update    Accounts
set       balance = :ybal + 100
where     accountId = 2;
                                        begin transaction;
                                        select    balance into :ybal2
                                        from      Accounts
                                        where     accountId = 2;
                                        select    balance into :xbal2
                                        from      Accounts
                                        where     accountId = 1;
update     Accounts
set        balance = :xbal - 100
where      accountId = 1;
commit;
                                        update     Accounts
                                        set        balance = :xbal2 + 100
                                        where      accountId = 1;
                                        update     Accounts
                                        set        balance = :ybal2 - 100
                                        where      accountId = 2;
                                        commit;
```

Serializable Transaction Executions

- Consider a set of transactions $S = \{T_1, \cdots, T_n\}$
- An execution of S is serializable if it is equivalent to some serial execution of S
- Let E_1 & E_2 denote two executions of S
- E_1 and E_2 are equivalent executions of S if
  1. both executions produce the same final database state, &
  2. both executions retrieve the same values: for every value read by some T_i in E_1, the corresponding read by T_i in E_2 returns the same value
- **Serializable executions** guarantee correctness of transaction executions

Interleaved Executions: Example 1

An interleaved execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select    balance into :xbal
from      Accounts
where     accountId = 1;
select    balance into :ybal
from      Accounts
where     accountId = 2;
update    Accounts
set       balance = :ybal + 100
where     accountId = 2;
                                              begin transaction;
                                              select    balance into :ybal2
                                              from      Accounts
                                              where     accountId = 2;
                                              select    balance into :xbal2
                                              from      Accounts
                                              where     accountId = 1;
update     Accounts
set        balance = :xbal - 100
where      accountId = 1;
commit;
                                              update     Accounts
                                              set        balance = :xbal2 + 100
                                              where      accountId = 1;
                                              update     Accounts
                                              set        balance = :ybal2 - 100
                                              where      accountId = 2;
                                              commit;
```

Non-Serializable Executions: Example 1a

Transfer(1,2,100)	Transfer(2,1,100)	Comments
begin transaction;		Accounts: (1,1000), (2,2000)

select balance **into** :xbal
from Accounts
where accountId = 1;
		xbal = 1000

select balance **into** :ybal
from Accounts
where accountId = 2;
		ybal = 2000

update Accounts **set** balance = :ybal + 100 **where** accountId = 2;		Accounts: (1,1000), (2,2100)
	begin transaction;	

	select balance **into** :ybal2
	from Accounts
	where accountId = 2;
		ybal2 = 2100

	select balance **into** :xbal2
	from Accounts
	where accountId = 1;
		xbal2 = 1000

update Accounts **set** balance = :xbal - 100 **where** accountId = 1;		Accounts: (1,900), (2,2100)
commit;		
	update Accounts **set** balance = :xbal2 + 100 **where** accountId = 1;	Accounts: (1,1100), (2,2100)
	update Accounts **set** balance = :ybal2 - 100 **where** accountId = 2;	Accounts: (1,1100), (2,2000)
	commit;	

Non-Serializable Executions: Example 1b

Transfer(1,2,100)	Transfer(2,1,100)	Comments
begin transaction;		Accounts: (1,1000), (2,2000)

select balance **into** :xbal
from Accounts
where accountId = 1;
		xbal = 1000

select balance **into** :ybal
from Accounts
where accountId = 2;
		ybal = 2000

update Accounts **set** balance = :ybal + 100 **where** accountId = 2;		Accounts: (1,1000), (2,2100)
	begin transaction;	

	select balance **into** :ybal2
	from Accounts
	where accountId = 2;
		ybal2 = 2000

	select balance **into** :xbal2
	from Accounts
	where accountId = 1;
		xbal2 = 1000

update Accounts **set** balance = :xbal - 100 **where** accountId = 1;		Accounts: (1,900), (2,2100)
commit;		
	update Accounts **set** balance = :xbal2 + 100 **where** accountId = 1;	Accounts: (1,1100), (2,2100)
	update Accounts **set** balance = :ybal2 - 100 **where** accountId = 2;	Accounts: (1,1100), (2,1900)
	commit;	

Interleaved Executions: Example 2

Another interleaved execution of Transfer(1,2,100) & Transfer(2,1,100)

```
begin transaction;
select      balance into :xbal
from        Accounts
where       accountId = 1;
select      balance into :ybal
from        Accounts
where       accountId = 2;
update      Accounts
set         balance = :ybal + 100
where       accountId = 2;
                                              begin transaction;
                                              select      balance into :ybal2
                                              from        Accounts
                                              where       accountId = 2;
update       Accounts
set          balance = :xbal - 100
where        accountId = 1;
commit;
                                              select       balance into :xbal2
                                              from         Accounts
                                              where        accountId = 1;
                                              update       Accounts
                                              set          balance = :xbal2 + 100
                                              where        accountId = 1;
                                              update       Accounts
                                              set          balance = :ybal2 - 100
                                              where        accountId = 2;
                                              commit;
```

Serializable Executions: Example 2a

Transfer(1,2,100)	Transfer(2,1,100)	Comments
begin transaction;		Accounts: (1,1000), (2,2000)

select balance **into** :xbal
from Accounts
where accountId = 1;
		xbal = 1000

select balance **into** :ybal
from Accounts
where accountId = 2;
		ybal = 2000

update Accounts **set** balance = :ybal + 100 **where** accountId = 2;		Accounts: (1,1000), (2,2100)
	begin transaction;	

	select balance **into** :ybal2
	from Accounts
	where accountId = 2;
		ybal2 = 2100

update Accounts **set** balance = :xbal - 100 **where** accountId = 1;		Accounts: (1,900), (2,2100)
commit;		

	select balance **into** :xbal2
	from Accounts
	where accountId = 1;
		xbal2 = 900

	update Accounts **set** balance = :xbal2 + 100 **where** accountId = 1;	Accounts: (1,1000), (2,2100)
	update Accounts **set** balance = :ybal2 - 100 **where** accountId = 2;	Accounts: (1,1000), (2,2000)
	commit;	

Non-Serializable Executions: Example 2b

Transfer(1,2,100)	Transfer(2,1,100)	Comments
begin transaction;		Accounts: (1,1000), (2,2000)

select balance **into** :xbal
from Accounts
where accountId = 1;
		xbal = 1000

select balance **into** :ybal
from Accounts
where accountId = 2;
		ybal2 = 2000

update Accounts **set** balance = :ybal + 100 **where** accountId = 2;		Accounts: (1,1000), (2,2100)
	begin transaction;	

	select balance **into** :ybal2
	from Accounts
	where accountId = 2;
		ybal2 = 2000

update Accounts **set** balance = :xbal - 100 **where** accountId = 1;		Accounts: (1,900), (2,2100)
commit;		

	select balance **into** :xbal2
	from Accounts
	where accountId = 1;
		xbal2 = 900

	update Accounts **set** balance = :xbal2 + 100 **where** accountId = 1;	Accounts: (1,1000), (2,2100)
	update Accounts **set** balance = :ybal2 - 100 **where** accountId = 2;	Accounts: (1,1000), (2,1900)
	commit;	

Quiz 1

Is the the following execution of transactions T_1 & T_2 serializable?

- T_1: Withdraw $100 from accountId 1
- T_2: Withdraw $500 from accountId 1

```
begin transaction;
select balance into :xbal from Accounts
where accountId = 1;
                                                begin transaction;
                                                select   balance into :xbal from Accounts
                                                where    accountId = 1;

                                                update   Accounts
                                                set      balance = :xbal - 500
                                                where    accountId = 1;
                                                commit;
update Accounts
set balance = :xbal - 100
where accountId = 1;
commit;
```

Quiz 2

Assume that the balances for accountId 1 and accountId 2 are \$1000 & \$2000, respectively. Consider a concurrent execution of transactions T_3 and T_4 where the values read by T_4 are \$1000 & \$2100. Is this concurrent execution serializable?

T_3:

```
begin transaction;
update Accounts
set balance = balance + 100
where accountId = 2;

update Accounts
set balance = balance - 100
where accountId = 1;
commit;
```

T_4:

```
begin transaction;
select balance
from Accounts
where accountId = 1;

select balance
from Accounts
where accountId = 2;
commit;
```

Quiz 3

Is the following execution of T_5 & T_6 serializable?

T_5	T_6	**Comments**
begin transaction;		Accounts: (1,1000), (2,2000)

select balance **into** :ybal
from Accounts
where accountId = 2;
		ybal = 2000

	begin transaction;	

	select balance **into** :xbal2
	from Accounts
	where accountId = 1;
		xbal2 = 1000

	select balance **into** :ybal2
	from Accounts
	where accountId = 2;
		ybal2 = 2000

update Accounts
set balance = :ybal + 200;
where accountId = 2;
		Accounts: (1,1000), (2,2200)

commit;		
	update Accounts **set** balance = :xbal2 + 100 **where** accountId = 1;	Accounts: (1,1100), (2,2200)
	commit;	

Quiz 4

Is the following execution of T_5, T_6 & T_7 serializable?

T_5	T_6	T_7	**Comments**
begin transaction;			Accounts: (1,1000), (2,2000)

select balance **into** :ybal
from Accounts
where accountId = 2;
			ybal = 2000

	begin transaction;		

	select balance **into** :xbal2
	from Accounts
	where accountId = 1;
			xbal2 = 1000

	select balance **into** :ybal2
	from Accounts
	where accountId = 2;
			ybal2 = 2000

update Accounts
set balance = :ybal + 200;
where accountId = 2;
			Accounts: (1,1000), (2,2200)

commit;			
		begin transaction;	

		select balance
		from Accounts
		where accountId = 1;
			1000

		select balance
		from Accounts
		where accountId = 2;
			2200

		commit;	
	update Accounts **set** balance = :xbal2 + 100 **where** accountId = 1;		Accounts: (1,1100), (2,2200)
	commit;		

ANSI SQL Isolation Levels

- The isolation level for a transaction affects what the transaction will read
- Dirty read = read value is produced by a transaction that has not yet committed
- Non-repeatable read = successive reads of the same tuple yield different values
- Phantom read = successive reads of a set of tuples satisfying a predicate yield different values

Dirty Read: Example

Consider the following execution of transactions T_1 & T_2

- T_1: Transfer(1,2,100)
- T_2: Read the balances for accountIds 1 & 2

T_1	T_2	**Comments**
begin transaction;		Accounts: (1,1000), (2,2000)

select balance **into** :xbal
from Accounts
where accountId = 1;
		xbal = 1000

select balance **into** :ybal
from Accounts
where accountId = 2;
		ybal = 2000

update Accounts **set** balance = :ybal + 100 **where** accountId = 2;		Accounts: (1,1000), (2,2100)
	begin transaction;	

	select balance
	from Accounts
	where accountId = 1;
		1000

	select balance
	from Accounts
	where accountId = 2;
		2100 **(dirty read!)**

	commit;	
update Accounts **set** balance = :xbal - 100 **where** accountId = 1;		Accounts: (1,900), (2,2100)
commit;		

Non-Repeatable Read: Example

Consider the following execution of transactions T_1 & T_2

- T_1: Transfer \$100 from accountId 1 to accountId 2
- T_2: Two successive reads of the balance of accountId 2

T_1	T_2	**Comments**
	begin transaction;	Accounts: (1,1000), (2,2000)

	select balance
	from Accounts
	where accountId = 2;
		2000

begin transaction;		
update Accounts **set** balance = balance + 100 **where** accountId = 2;		Accounts: (1,1000), (2,2100)
update Accounts **set** balance = balance - 100 **where** accountId = 1;		Accounts: (1,900), (2,2100)
commit;		

	select balance
	from Accounts
	where accountId = 2;
		2100 **(non-repeatable read!)**

	commit;	

Phantom Read: Example

Consider the following execution of transactions T_1 & T_2

- T_1: Two successive reads of tuples with balance more than \$1000.
- T_2: Insert a new tuple with balance more than \$1000.

T_1	T_2	**Comments**
	begin transaction;	Accounts: (1,100), (2,2000)

	select accountId
	from Accounts
	where balance $>$ 1000;
		$\{2\}$

insert into Accounts **values** (3, 3000);		Accounts: (1,100), (2,2000), (3, 3000)

	select accountId
	from Accounts
	where balance $>$ 1000;
		$\{2, 3\}$ **(phantom read!)**

	commit;	

ANSI SQL Isolation Levels

- ANSI SQL defines four isolation levels
- Configure using **set transaction isolation level** statement

Isolation Level	Dirty Read	Non-repeatable Read	Phantom Read
READ UNCOMMITTED	possible	possible	possible
READ COMMITTED	not possible	possible	possible
REPEATABLE READ	not possible	not possible	possible
SERIALIZABLE	not possible	not possible	not possible

Comments on SQL Isolation Levels

- In many DBMSs, running transactions at SQL's `SERIALIZABLE` isolation level guarantees that the execution is serializable
- However, depending on the application's transaction workload, a non-serializable execution could still be correct
- Ideally, use the weakest isolation level to obtain correct transaction executions for application

Transaction Example: Seat Booking

- Customers (custId, name, · · ·)
- Seats (seatNumber, price)
- Bookings (eventDate, seatNumber, custId)

```
begin transaction;
-- Customer 123 wants to book a seat for June 20 2018

-- Display available seats
select seatNumber from Seats
where seatNumber not in
    (select seatNumber from Bookings
    where eventDate = '2018-06-20');

-- Customer 123 books seat 10
insert into Bookings values ('2018-06-20', 10, 123);
commit;
```

Transaction Example: Seat Booking

- T_1: Customer 123 books seat 10 for 2018-06-20
- T_2: Customer 456 books seat 20 for 2018-06-20

```
begin transaction;

set transaction isolation level read committed;

select seatNumber from Seats
where seatNumber not in (
        select seatNumber from Bookings
        where eventdate = ’2018-06-20’);
                                                  begin transaction;

                                                  set transaction isolation level read committed;

                                                  select seatNumber from Seats
                                                  where seatNumber not in (
                                                          select seatNumber from Bookings
                                                          where eventdate = ’2018-06-20’);
insert into Bookings values (’2018-06-20’,10,123);
commit;
                                                  insert into Bookings values (’2018-06-20’,20,456);
                                                  commit;
```