Assignments 4 + 5

Write-Ahead-Log (WAL)

1. Return the current LSN of your WAL file

2. Run the transaction on next slide. (Make sure that the Pks do NOT yet exist in your table lesson.)

3. Return the current LSN of your WAL file

4. Dump the WAL from the first to the second LSN

    pg_waldump -s start_LSN -e end_LSN -p "path to WAL directoy"
    path usually is "...\PostgreSQL\16\data\pg_wal"

    Attention: pg_waldump is NOT a PSQL command but a separate tool. Simply run it from a terminal.

eva.knirsch@kiu.edu.ge

WAL

```
Begin;
update student set s_balance = s_balance - 3 where s_username= 'Rose';
update teacher set t_payment = t_payment + 3 where t_id = 1;
select t_payment from teacher where t_id = 1;
INSERT INTO lesson(t_id, lesson_time, s_username, subjectcode)
        VALUES (1, '2024-03-03 05:22:12.000000', 'Rose', 'EN');
INSERT INTO lesson(t_id, lesson_time, s_username, subjectcode)
        VALUES (1, '2024-03-04 05:22:12.000000', 'Rose', 'EN');
INSERT INTO lesson(t_id, lesson_time, s_username, subjectcode)
        VALUES (1, '2024-03-05 05:22:12.000000', 'Rose', 'EN');
Commit;
```


eva.knirsch@kiu.edu.ge

WAL

How to interpret the WAL records:

1. You should see the two updates
   - probably as HOT_Updates (will explain next lecture)
   - the old_xmax was the transaction id of the transaction that deleted the old version
     (Verify that your current txid is one more incremented.)
   - the old_off is the offset of the old row,
     the new_Xmax:0 indicates that the new version is the valid version
   - the new_off is the offset of the new row version. Verify by displaying the ctid.

2. You should see the 3 inserts into the heap file
   - verify with a <select *, ctid, xmin, xmax, from lesson> the block, offsets and txid
   - for each insert into the heap file you see inserts into 2 index files (Btrees), into the b-tree-leaves
     - why inserts into 2 different btrees?
     - why inserts into the btree leaves?

3. At last, you should see your transaction commit

eva.knirsch@kiu.edu.ge

Dirty Read Anomaly in PostgreSQL?

```
/* Applying a dirty read scenario to our course example */
/* Session 1: dirty read ?  */
Begin;
drop table if exists transaction_log;
CREATE TEMPORARY TABLE transaction_log (message_text varchar(50), t_payment_value INT);
INSERT INTO transaction_log (message_text, t_payment_value) select 'Payment_amount',t_payment FROM teacher
WHERE t_id = 1;
UPDATE teacher SET t_payment = 0 WHERE t_id = 1;
SELECT pg_sleep(30);
INSERT INTO transaction_log (message_text, t_payment_value) select 'after_reset_balance',t_payment FROM teacher
WHERE t_id = 1;
rollback;
```


eva.knirsch@kiu.edu.ge

Dirty Read Anomaly in PostgreSQL?

```
/* Session 2: runs during session 1 pg_sleep - dirty read ? */
Begin;
drop table if exists transaction_log;
CREATE TEMPORARY TABLE transaction_log (t_payment_value INT);
INSERT INTO transaction_log (t_payment_value) select t_payment FROM teacher WHERE t_id = 1;
COMMIT;
```

Consistent result would be:
Inconsistent result ("dirty read") would be:

Does it make any difference for the read in session 2 if the transaction in session 1 commits or rollbacks?

eva.knirsch@kiu.edu.ge

Non-Repeatable Read Anomaly

Example:
T1: Returns the student names where s_balance < 5
T2 Adds 5 to s_balance of student

1	T1	T2	Result
2	Begin		
3	Select s_name where s_balance < 5	Begin	T1 returns the names of x students, among them student y
5		Update student set s_balance = s_balance + 5 where s_name = y	T2 **updates** s_balance of student y, s-balance of y now > 5
6		**commit**	
7	Select s_name where s_balance < 5		What does T1 return? Is student y still in the result set?
8	Commit		

Run the transactions also in isolation level Repeatable Read

Phantom Read Anomaly in PostgreSQL?

Example:
T1: Counts the teachers with t_payment <=5
T2 Inserts a new teacher with default t_payment = 0

1	T1	T2	Result
2	BoT		
3	select count(*) FROM teacher WHERE t_payment <=5;	BoT	T1 returns number x
4		Insert into teacher ...	T2 inserts a new teacher with t_payment = 0
5		commit	
6	select count(*) FROM teacher WHERE t_payment <=5;		T1 returns x+1, a "phantom row"
7	Commit		

Run the transactions in isolation level Repeatable Read

2PL

Given is the following schedule. How would it work under2PL?

T1:Read(A) T2:Read(A) T1:Write(A) T2:Write(A) T1:Read(B) T2:Read(B) T1:Write(B) T2:Write(B)

What is the result?

Operation	T1	T2
1	Begin	
2		
3		
4		
5		
6		
7		
8		
9		
10		

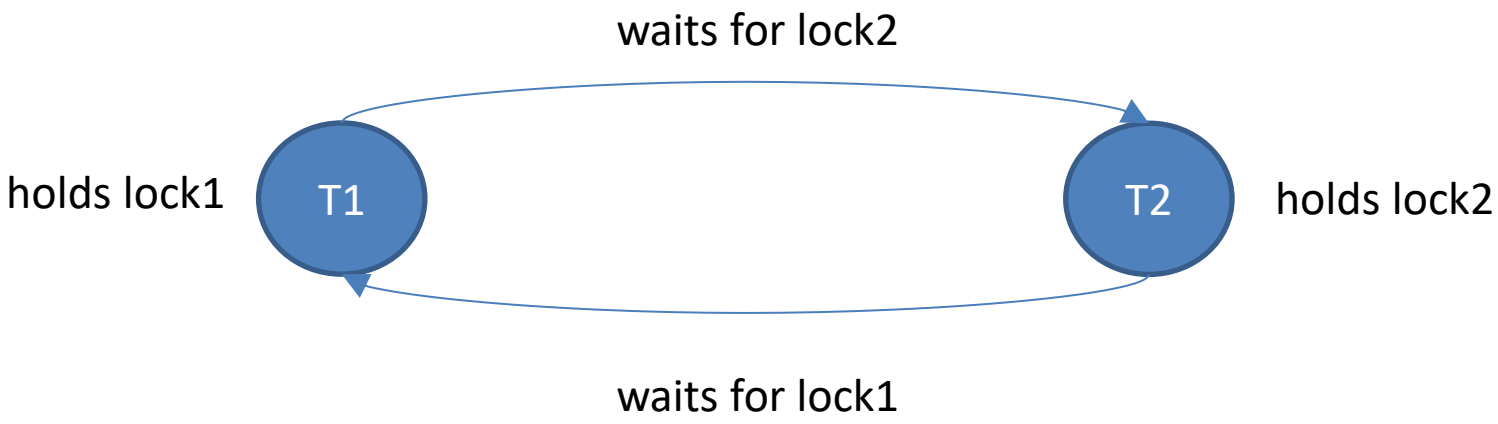
eva.knirsch@kiu.edu.ge

Deadlock

1	T1	T2	T3
	Begin		
2	subtract 3 from s_balance of student Mickey	Begin	
3	application logic pg_sleep(40)	Add 3 to t_id = 1	Begin
4		application logic pg_sleep(35)	subtract 3 from s_balance of student Rose
5			application logic pg_sleep(30)
6	Add 3 to t_id = 1	subtract 3 from s_balance of student Rose	subtract 3 from s_balance of student Mickey
7	commit	commit	Add 6 to t_id = 2
8			commit

1. Write the code for T1, T2 and T3 and let them run parallel in 3 sessions.
2. Which transaction is rolled back, which commit in what sequence?
3. What is the error message?
4. Explain the problem and the result.

Waits-For-Graph



The database breaks the cycle by aborting one or more transactions. The transaction picked for abortion is called the victim. The decision which transaction is the victim depends on the database system or database settings.

Commonly one of the following methods is applied to pick the vistim transaction:

- youngest transaction is rolled back
- transacton that holds fewest locks is rolled back (→ minimizes rollback effort)

Which "victim method picking" did you observe running the deadlock simulation assignments of slide before?

eva.knirsch@kiu.edu.ge

MVCC

Given is the following schedule s:

T1 read(a) T1write(a) T2read(a) T2write(b) T1 read(b) T1write(c)

Assume that all writes are updates. Objects a,b,a nd c already exist.
How does the execution under MVCC look like? Do the two transactiobn run through?

T10	T11
Begin	

Version	Value	Created_ by	Deleted _by

eva.knirsch@kiu.edu.ge

MVCC

Given is the following schedule s:

T1 read(a) T1 write(a) T2 read(a) T2 write(b) T1 read(b) T1 write (c)

Assume that object a and b exist. T2 write(b) is a delete and T1 write (c) is an insert. How does the execution under MVCC look like?

T1	T2
Begin	

Version	Value	Created_ by	Deleted _by

eva.knirsch@kiu.edu.ge

Write Skew anomaly

1	T1	T2	Result
2	Begin		2 persons must be on duty
3	n_on_duty = select count(*) from personnel where on_duty=TRUE	Begin	T1 returns 3 persons on duty
4		n_on_duty = select count(*) from personnel where on_duty=TRUE	T2 returns 3 persons on duty
5	if (n_on_duty > 2) (update personnel set on_duty = False where name= "Alice")		update → 2 persons on duty
6		if (n_on_duty > 2) (update personnel set on_duty = False where name= "Bob")	update → ?
7	commit;		result?
8		Commit;	result?

- Create a table and populate the table so that it implements the above use case
- Code the above write skew in 2 Postgres transactions / sessions
- Run the two transactions once in isolation level Repeatable Read and once in Serializable.