	Course	**Databases and Information Systems 2020**		
	Exercise Sheet	**3**		
	Points	–		
	Release Date	**May 12th 2020**	Due Date	**May 19th 2020**

3 Synchronization with Locking Protocols

Note

- Include SQL commands in your report and not as separate SQL files.

3.1 Isolation Levels and SQL

a) Open a connection to database `dis-2020`[1] with a tool of your choice. If you prefer, you can use a local PostgreSQL installation.

- How can you determine the currently set isolation level?
- What is the default isolation level of PostgreSQL?
- How can the isolation level be changed during a session in PostgreSQL?

Provide a solution applicable to all kinds of tools. Avoid tool specific shortcuts or QoL features. *Hint: Consider checking the PostgreSQL documentation.*

b) Write down a SQL code for creating a simple table `OPK` with the columns `ID` and `NAME`. No column is declared as a primary key.

c) Write down a SQL command suitable for filling OPK with data. Afterward, the data should look like this:

OPK

ID	NAME
1	shaggy
2	fred
3	velma
4	scooby
5	daphne

d) Consider an open connection to an arbitrary database like DB2[2] with a current transaction level of **Read Committed** (PostgreSQL uses a Multiversion Concurrency Control mechanism, which makes this question not applicable). The connection executes the following SQL commands:

```
-- Begin Transaction
START TRANSACTION;
-- Query one row from table OPK
SELECT * FROM OPK WHERE ID = 3;
-- ! Check for current locks !
COMMIT;
```

Discuss what locks you would expect are held at this point (and before a commit)?
Hint: You can use Slides 8-11 from the Additional Slides for this Exercise Sheet. Caution: The Slides were originally written for the database DB2.

e) Discuss what locks you would expect are held if the SQL command from d) is done with the isolation level `REPEATABLE READ`.

[1] Database from sheet 1 & 2
[2] Consider looking into the Additional Slides (Slides 8-11).

3.2 Lock Conflicts

a) Given the following scenario: You have two open connections. Connection 1 (isolation level *Repeatable Read*) queries the rows with `ID>3` from OPK but does not commit yet. Connection 2 (isolation level *Committed Read*) adds a new row to the table that satisfies the selection predicate.

- What happens? What is the output of Connection 1?
- Compare the state before the transactions with the state after the transactions.
- What can be observed if Connection 1 commits and execute its SQL command again?
- Can we observe a Canonical Synchronization Problem? If yes, explain which one and why it appears.

T_1 (Repeatable Read)	T_2 (Committed Read)
`START TRANSACTION;`	

`SELECT *`
`FROM OPK`
`WHERE ID > 3;`	

	`START TRANSACTION;`
	`INSERT INTO OPK` `VALUES(6, 'scrappy');`
	`COMMIT;`
`COMMIT;`	

b) Repeat the scenario from a) with the isolation level of Connection 1 set to *Serializable*. Now repeat the steps from a).

- Want can be observed now? What is the output of Connection 1?
- Compare the state before the transactions with the state after the transactions.
- Can we observe a Canonical Synchronization Problems? If yes, explain which one and why it appears.

c) Given the following scenario: You have two open connections. Connection 1 (isolation level *Serializable*) queries one row from table `OPK` (selection via the ID attribute, ID=3) but does not commit yet. Connection 2 (isolation level *Committed Read*) change another row's `NAME` value.

- In this scenario Connection 2 has to wait until Connection 1 commits. Explain why.
- Discuss, what lock can be potentially encountered on the table OPK? Which Connection do the locks belong to?

T_1 (Serializable)	T_2 (Committed Read)
`START TRANSACTION;`	

`SELECT *`
`FROM OPK`
`WHERE ID = 3;`	

	`START TRANSACTION;`

	`UPDATE OPK`
	`SET NAME = 'norville'`
	`WHERE id = 1;`

	`COMMIT;`
`COMMIT;`	

d) Write down a SQL code for creating a second table `MPK`, again with columns `ID` and `Name`, and declare `ID` a primary key. The new table should be filled with the same initial data as the first table `OPK`. The SQL code for filling `MPK` has to be added, too.

e) The transactions from c) are repeated with the table `MPK`. This time, Connection 2 doesn't have to wait for the commit of Connection 1. Explain the behavior.