Module 1

Exercise 1: A look at PostgreSQL pages

1. Connect to your instance and create a database
2. Create a table containing only one `VARCHAR` column, insert `'Hello World!'` in this table.
   - ⚠ do not forget to `CHECKPOINT`!
   - What did this achieve?
3. Query the `pg_database` table to find the OID (Object Identifier) of your database
4. Exec into your database container. Go to `/var/lib/postgresql/data/base/` and find the folder containing your database. What is inside?
5. Find the file for your table using `pg_relation_filepath()`. Use the `du -sb` command to check its size in bits. What is this size? Why?
6. Install `hexdump` in the container using `apt update && apt install bsdmainutils`
7. Run `hexdump -C` on your table file. What do you see?
8. Insert another value in the table. Check again.
   - ⚠ do not forget to `CHECKPOINT`!
9. Same question with a `NULL` value
10. Now delete the whole table and check again. What is happening? How could you fix this?
11. Run `VACUUM FULL`on your table. The file has disappeared! Find the new file name, and look at it.

Exercise 2: Tuple Storage and Column Padding

We are going to use two pretty handy - but quite unknown - functions named `row()` and `pg_column_size()`.

- `pg_column_size()` returns **how much space in bytes** the supplied value would occupy on disk. For example, `pg_column_size(1::int4)` will suprisingly return 4, as a 4-bytes int will occupy 4 bytes.
- `row()` builds a tuple. For example `row(1::int2, 'a'::char)` builds the tuple (1, 'a')

1. Using both functions, compute the size of an **empty row**.
   - What is this size, What could be explain this number?
2. What is the size of a row made of 8 `NULL` values of any type? And nine `NULL` values?
   - What could be happening?
3. Could you use the functions to compute the size of the a row from the following table

```
CREATE TABLE test
(
  foo int2,
  toto int8,
  bar int2,
  tata int8,
  baz int2,
```

```
  titi int8
);
```

4. How would you reorganize the rows in the above table to limit the storage size. What is the minimum size you can reach with those columns?

Exercise 3: The Write-Ahead Log (WAL)

1. Connect to your instance and create a database or use an existing one
2. Install the extension `pg_walinspect` with `CREATE EXTENSION`
3. Create a table of any structure
4. Write down the current LSN with `SELECT pg_current_wal_lsn()`
    - You can use `'FFFFFFF/FFFFFFF'` as second argument
5. Insert a few lines for example.
6. Use the `pg_get_wal_records_info()` function to get all the WAL records between the two LSNs you noted.
7. Now, run the following query

```
SELECT encode(block_data,'escape')
FROM pg_get_wal_block_info(<lsn>,'FFFFFFF/FFFFFFF');
```

8. What do you see?

Exercise 4: Transactions Isolation

1. Create a simple table named `test` with two INT columns named `id` and `value`
2. Open two tabs in `pgadmin`. We will cal them tab1 and tab2 from now on
3. In tab1, run `BEGIN` to start a transaction. In this same tab, run an `INSERT` command to add a new line. Finally run a `SELECT *` from the table in both tab1 and tab2.
    - What do you see?
    - Why is that? What is the isolation level you are using?
4. `COMMIT` the transaction in tab1. Run again the `SELECT` in both tabs. What happened?
5. Now in tab1 start a new transaction with `BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;`
6. `SELECT` the content of the table. Go to the second tab and add a new record. Go back to the first one and `SELECT` again.
    - What do you see?
7. `ROLLBACK`the transaction in tab1.
8. Finally now use `BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;` in tab1. Run an update on one of the lines in tab2. Now run the same exact UPDATE in tab1. What is happening?

Exercise 5: Explicit locking

1. Create a new table of any form. Insert a few records in this table.
2. Start a transaction in a tab with `BEGIN`. Issue a

```
SELECT *
FROM <table>
FOR UPDATE
```

3. Go to a second tab. Run the following:

```
SELECT l.*
FROM pg_locks l
JOIN pg_class t ON l.relation = t.oid
WHERE t.relkind = 'r'
  AND t.relname = '<table>';
```

- What do you see?

4. Now in the same tab, try to modify the table, for example run an update? What happens?
5. Go back to the first tab and `ROLLBACK` the transaction. What happened in the second tab?

Go to https://www.postgresql.org/docs/current/explicit-locking.html to understand a bit more about what you just did.