

Track: B20-AI-01	Week 2
Name: Artem Chernitsa	Introduction to Big Data
Email: a.chernitsa@innopolis.university	28.03.2023

## Report

### <PostgreSQL Part>

#### Task 1

##### Your query as a text

**SELECT** c.user\_id, p.venue\_id, p.category, c.utc\_time + c.timezone\_offset\_mins **as** local\_datetime

**FROM** checkins c **LEFT JOIN** pois p **ON** c.venue\_id = p.venue\_id

**WHERE** p.country = 'RU';

The screenshot shows the DBeaver SQL Editor interface. The SQL Editor window displays a query that filters check-ins for users in Russia. The query is as follows:

```

1 /*[4 point] Which are the user ids, venue ids, venue categories, local datetime
2 * of the check-ins in Russia (Country code RU)?*/
3 SELECT c.user_id, p.venue_id, p.category, c.utc_time + c.timezone_offset_mins as local_datetime
4 FROM checkins c LEFT JOIN pois p ON c.venue_id = p.venue_id
5 WHERE p.country = 'RU';
6
7
8 /*[4 point] What are the friend ids of users who checked in Russia and have
9 * friendship before April 2012?*/
10 SELECT COUNT(*) FROM friendship_before fb;
11
12 SELECT friends.user_id, friends.friend_id FROM
13 (SELECT fb.user_id, fb.friend_id FROM
14 FROM friendship_before fb
15 UNION
16 SELECT fb2.friend_id, fb2.user_id
17 FROM friendship_before fb2) friends
18 ORDER BY friends.user_id, friends.friend_id;
19
20 SELECT friends.user_id, friends.friend_id FROM
21 (SELECT fa.user_id, fa.friend_id
22 FROM friendship_after fa
23 UNION
24 SELECT fa2.friend_id, fa2.user_id
25 FROM friendship_after fa2) friends
26 ORDER BY friends.user_id, friends.friend_id;
27

```

The Results window shows the output of the first query, displaying columns: user\_id, venue\_id, category, and local\_datetime. The results are as follows:

user_id	venue_id	category	local_datetime
1,412,323	4bde5c0e0e3a593a0d230b0	Movie Theater	2012-04-03 22:11:35.000
1,412,323	4debd28ab0fb82937f8d84e	Other Great Outdoors	2012-04-04 01:20:18.000
1,265,452	4f0480cc0e61e937cf185f0a	Home (private)	2012-04-04 01:48:13.000
909,429	4d6810ce709bb60cd69eb114	Home (private)	2012-04-04 07:02:31.000
909,429	4d6e34bf2427224bc8b6c94d	Building	2012-04-04 08:10:17.000
909,429	4d65e5fe153ba0939b3ae4f8	Office	2012-04-04 08:47:18.000
1,642,244	4d2d28aacde68eac27a6c998	Gym	2012-04-04 12:44:21.000
1,783,104	4c0c6b85d64c0f471a47255d	Park	2012-04-04 08:03:56.000
1,228,918	4d7a1e78c307a35df534f31	Office	2012-04-04 09:21:05.000
1,967,821	4de8175b88774aa92c96bbf0	School	2012-04-04 09:26:06.000
1,158,601	4c6cd8d7fe77bfb7f74a023	University	2012-04-04 09:28:07.000
751,379	4d6b766c92f6b60c7ed6a0e0	Student Center	2012-04-04 10:01:00.000
751,379	4f7be8f4e4b086fa1eb09e14	College Auditorium	2012-04-04 10:23:50.000

### Your comments (optional)

I did it on the HDP cluster, but I didn't save any screenshots. Since I dropped my server cause of mongoDB, I simply up docker container with postgres and make absolutely the same just to save screenshots.

## Task 2

### Your query as a text

```
SELECT DISTINCT fa.friend_id FROM

    (SELECT fa.user_id, fa.friend_id

FROM friendship_after fa

UNION

SELECT fa2.friend_id, fa2.user_id

FROM friendship_after fa2) fa

INNER JOIN

    (SELECT fb.user_id, fb.friend_id

FROM friendship_before fb

UNION

SELECT fb2.friend_id, fb2.user_id

FROM friendship_before fb2) fb

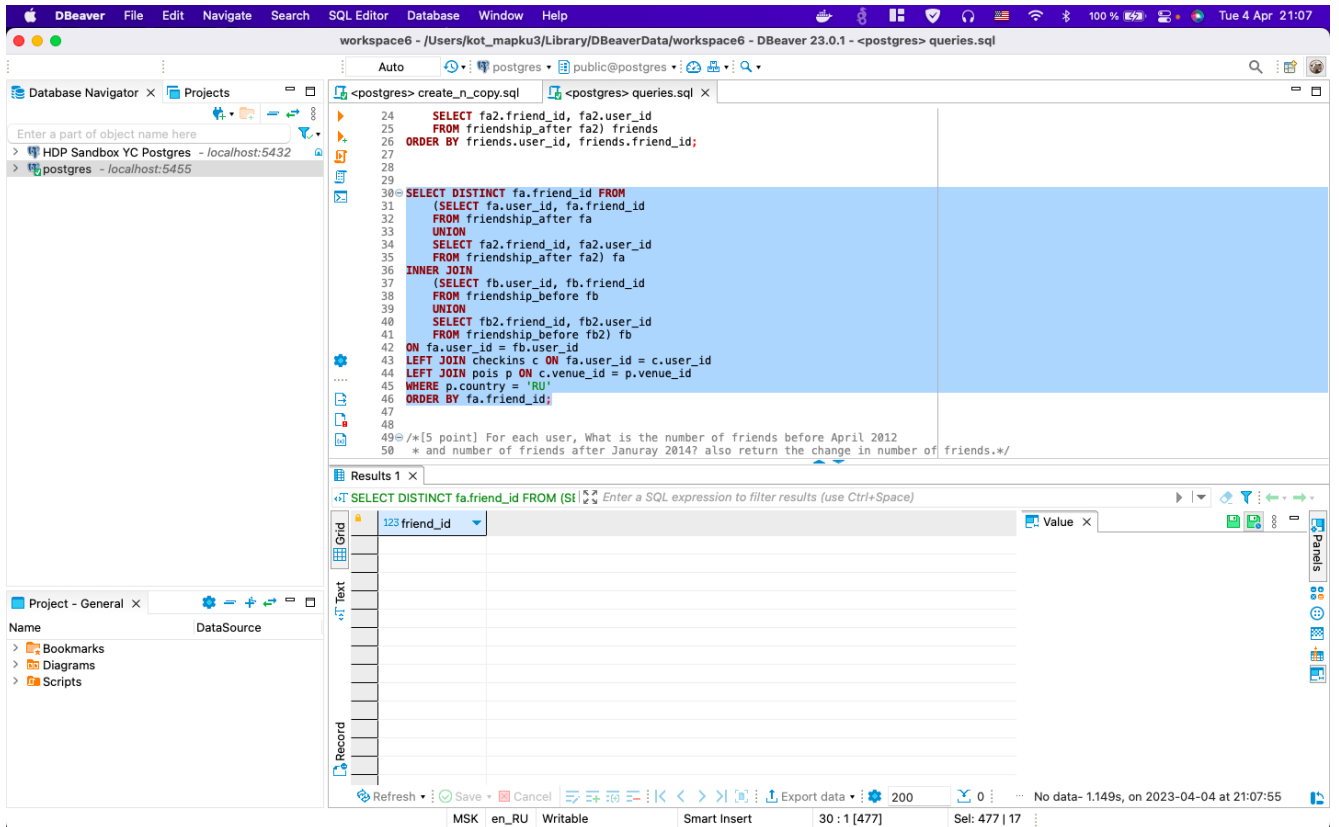
ON fa.user_id = fb.user_id

LEFT JOIN checkins c ON fa.user_id = c.user_id

LEFT JOIN pois p ON c.venue_id = p.venue_id

WHERE p.country = 'RU'

ORDER BY fa.friend_id;
```



### Your comments (optional)

For RU code is empty, but e.g. for the US is gives a lot of rows as a result.

## Task 3

### Your query as a text

```

SELECT u.user_id, u.fa, u.fb, u.fa - u.fb AS difference FROM (
    SELECT u.user_id, SUM(u.num_fb) AS fb, SUM(u.num_fa) AS fa FROM (
        SELECT fb.user_id, fb.num_fb, fb.num_fa FROM (
            SELECT fb.user_id, COUNT(fb.user_id) AS num_fb, 0 AS num_fa
            FROM
                (SELECT fb.user_id, fb.friend_id
                 FROM friendship_before fb
                 UNION
                 SELECT fb2.friend_id, fb2.user_id

```

```

        FROM friendship_before fb2

    ) fb

    GROUP BY fb.user_id

) AS fb

UNION

SELECT fa.user_id, fa.num_fb, fa.num_fa FROM (

    SELECT fa.user_id, 0 AS num_fb, COUNT(fa.user_id) AS num_fa

    FROM

        (SELECT fa.user_id, fa.friend_id

        FROM friendship_after fa

        UNION

        SELECT fa2.friend_id, fa2.user_id

        FROM friendship_after fa2

        ) fa

    GROUP BY fa.user_id

) AS fa

ORDER BY user_id

) AS u

GROUP BY user_id

) AS u;

```

The screenshot shows the DBeaver SQL Editor interface. The top menu bar includes File, Edit, Navigate, Search, SQL Editor, Database, Window, and Help. The main window displays a SQL query in the editor, which is a complex query involving multiple subqueries and joins to calculate the difference in friend counts for users before and after a change. The query is as follows:

```

SELECT fb.user_id, COUNT(fb.user_id) AS num_fb, 0 AS num_fa
FROM
  (SELECT fb.user_id, fb.friend_id
   FROM friendship_before fb
   UNION
   SELECT fb2.friend_id, fb2.user_id
   FROM friendship_before fb2
   ) fb
GROUP BY fb.user_id
) AS fb
UNION
SELECT fa.user_id, fa.num_fb, fa.num_fa FROM (
  SELECT fa.user_id, 0 AS num_fb, COUNT(fa.user_id) AS num_fa
  FROM
    (SELECT fa.user_id, fa.friend_id
     FROM friendship_after fa
     UNION
     SELECT fa2.friend_id, fa2.user_id
     FROM friendship_after fa2
    ) fa
  GROUP BY fa.user_id
) AS fa
ORDER BY user_id
) AS u
GROUP BY user_id

```

The results are displayed in a table with the following columns: user\_id, fa, fb, and difference. The first row shows user\_id 837 with fa=1 and fb=1, resulting in a difference of 0.

user_id	fa	fb	difference
837	1	1	0
3,365	2	2	0
4,489	1	1	0
6,133	1	1	0
6,863	1	1	0
7,570	3	2	1
9,437	1	1	0
10,383	1	1	0
11,522	1	1	0
12,330	1	1	0
13,477	0	2	-2
14,314	3	1	2
16,527	2	1	1

Your comments (optional)

## Task 4

Your query as a text

```

SELECT DISTINCT p.category FROM (
  SELECT fb.friend_id FROM (
    SELECT DISTINCT user_id FROM checkins c ORDER BY user_id ASC
  LIMIT 10 /* get first 10 users */
) AS u
RIGHT JOIN (
  SELECT fb.user_id, fb.friend_id
  FROM friendship_before fb
  UNION

```

**SELECT** fb2.friend\_id, fb2.user\_id

**FROM** friendship\_before fb2) fb **ON** fb.user\_id = u.user\_id

**EXCEPT**

**SELECT** u.user\_id **FROM** (

**SELECT DISTINCT** user\_id **FROM** checkins c **ORDER BY** user\_id **ASC**

**LIMIT 10** /\* left only friends of users \*/

) **AS** u

) **AS** u

**LEFT JOIN** checkins c **ON** u.friend\_id = c.user\_id /\* get all checkins for friends \*/

**LEFT JOIN** pois p **ON** c.venue\_id = p.venue\_id

**WHERE** p.country = 'RU';

The screenshot shows the DBeaver SQL Editor interface. The top menu bar includes File, Edit, Navigate, Search, SQL Editor, Database, Window, and Help. The main window displays a SQL query in the editor, which is a complex query involving multiple joins and subqueries. The query is highlighted in blue. Below the editor, the 'Results' panel shows the results of the query, which is a table with two columns: 'id' and 'category'. The table contains 43 rows of data, including categories like Airport, Automotive Shop, Bakery, Bank, Bookstore, Burger Joint, Caf, Capitol Building, Car Wash, Church, Coffee Shop, and College Academic Building. The status bar at the bottom indicates that 43 rows were fetched in 0.9s.

**Your comments (optional)**

## Task 5

Your query as a text

```
SELECT /*u.user_id, u.f1_id, u.f2_id,*/ DISTINCT u.f3_id FROM (

    SELECT u.user_id, u.f1_id, u.f2_id, u.f3_id FROM (

        SELECT u.user_id, f1.friend_id AS f1_id, f2.friend_id AS f2_id, f3.friend_id
        AS f3_id FROM (

            SELECT DISTINCT users.user_id FROM

                (SELECT fb.user_id, fb.friend_id

                    FROM friendship_before fb

                    UNION

                    SELECT fb2.friend_id, fb2.user_id

                    FROM friendship_before fb2) users

            LEFT JOIN checkins c ON c.user_id = users.user_id

            LEFT JOIN pois p ON c.venue_id = p.venue_id

            WHERE p.category = 'Post Office'

        ) AS u

        LEFT JOIN (

            SELECT fb.user_id, fb.friend_id

            FROM friendship_before fb

            UNION

            SELECT fb2.friend_id, fb2.user_id

            FROM friendship_before fb2

        ) AS f1 ON u.user_id = f1.user_id

        LEFT JOIN (

            SELECT fb.user_id, fb.friend_id

            FROM friendship_before fb

            UNION
```

```

        SELECT fb2.friend_id, fb2.user_id

        FROM friendship_before fb2

    ) AS f2 ON f1.friend_id = f2.user_id

    LEFT JOIN (

        SELECT fb.user_id, fb.friend_id

        FROM friendship_before fb

        UNION

        SELECT fb2.friend_id, fb2.user_id

        FROM friendship_before fb2

    ) AS f3 ON f2.friend_id = f3.user_id

    ORDER BY u.user_id ASC

) AS u

WHERE u.user_id != u.f1_id AND

        u.f1_id != u.f2_id AND

        u.f2_id != u.f3_id AND

        u.f3_id != u.f1_id AND

        u.user_id != u.f2_id AND

        u.user_id != u.f3_id

) AS u

CROSS JOIN (

    SELECT fb.user_id, fb.friend_id

    FROM friendship_before fb

    UNION

    SELECT fb2.friend_id, fb2.user_id

    FROM friendship_before fb2

) AS f

CROSS JOIN (

```



**SELECT** fb.user\_id, fb.friend\_id

**FROM** friendship\_before fb

**UNION**

**SELECT** fb2.friend\_id, fb2.user\_id

**FROM** friendship\_before fb2

) **AS** f2

**CROSS JOIN** (

**SELECT** fb.user\_id, fb.friend\_id

**FROM** friendship\_before fb

**UNION**

**SELECT** fb2.friend\_id, fb2.user\_id

**FROM** friendship\_before fb2

) **AS** f3

**WHERE**

u.user\_id = f.user\_id **AND** u.f2\_id = f.friend\_id **AND**

u.user\_id = f2.user\_id **AND** u.f3\_id = f2.friend\_id **AND**

u.f1\_id = f3.user\_id **AND** u.f3\_id = f3.friend\_id;

workspace6 - /Users/kot\_mapku3/Library/DBBeaverData/workspace6 - DBBeaver 23.0.1 - <postgres> queries.sql

Database Navigator × Projects

Enter a part of object name here

> HDP Sandbox YC Postgres - localhost:5432

> postgres - localhost:5455

```
151 ) AS u
152 CROSS JOIN (
153     SELECT fb.user_id, fb.friend_id
154     FROM friendship_before fb
155     UNION
156     SELECT fb2.friend_id, fb2.user_id
157     FROM friendship_before fb2
158 ) AS f
159 CROSS JOIN (
160     SELECT fb.user_id, fb.friend_id
161     FROM friendship_before fb
162     UNION
163     SELECT fb2.friend_id, fb2.user_id
164     FROM friendship_before fb2
165 ) AS f2
166 CROSS JOIN (
167     SELECT fb.user_id, fb.friend_id
168     FROM friendship_before fb
169     UNION
170     SELECT fb2.friend_id, fb2.user_id
171     FROM friendship_before fb2
172 ) AS f3
173 WHERE
174     u.user_id = f.user_id AND u.f2_id = f.friend_id AND
175     u.user_id = f2.user_id AND u.f3_id = f2.friend_id AND
176     u.f1_id = f3.user_id AND u.f3_id = f3.friend_id;
177
```

Results 1 ×

SELECT DISTINCT u.f3\_id FROM ( SELECT Enter a SQL expression to filter results (use Ctrl+Space)

	123 f3_id	Value
1	53,645	53645
2	173,784	
3	309,852	
4	406,437	
5	685,324	
6	748,322	
7	1,905,947	

Project - General ×

Name DataSource

Bookmarks

Diagrams

Scripts

Refresh Save Cancel Export data 200 7

MSK en\_RU Writable Smart Insert 177 : 1 [1965] Sel: 1965 | 68

Your comments (optional)

## <Neo4j Part>

### Task 1

#### Your schema output as a text

```
nodes
relationships

[(:User {name: "User", indexes: ["user_id"], constraints: []}), (:Venue
[[:FRIEND_BEFORE], [:FRIEND_AFTER], [:CHECK_IN]]
{name: "Venue", indexes: ["venue_id"], constraints: []})]
```

The screenshot displays the Neo4j Desktop application. The top bar shows the menu (File, Edit, View, Window, Help, Developer) and the current session name 'foursquare-neo4j'. The main workspace is divided into two sections. The top section shows the result of the command 'call db.schema.visualization()', displaying a graph with two nodes: a purple circle labeled 'User' and an orange circle labeled 'Venue'. They are connected by a 'CHECK\_IN' relationship. The 'User' node also has two self-loops labeled 'FRIEND...'. The bottom section shows the result of the Cypher query 'MATCH (u:User)←[:FRIEND\_AFTER]→(v), (u:User)←[:FRIEND\_BEFORE]→(v), (u:User)←[:CHECK\_IN]→(ven:...)'. The graph shows multiple 'User' nodes (purple circles) connected to 'Venue' nodes (orange circles) via 'FRIEND\_AFTER', 'FRIEND\_BEFORE', and 'CHECK\_IN' relationships. On the right side, there are two 'Overview' panels. The top panel shows 'Node labels' with 2 nodes (User 1, Venue 1) and 'Relationship types' with 3 types (FRIEND\_BEFORE 1, FRIEND\_AFTER 1, CHECK\_IN 1). The bottom panel shows 'Node labels' with 18 nodes (User 1, Venue 17) and 'Relationship types'.

#### Your comments (optional)

### Task 2

<b>Your query as a text</b>

< put here the screenshot of the output>

<b>Your comments (optional)</b>

### Task 3

<b>Your query as a text</b>

< put here the screenshot of the output>

<b>Your comments (optional)</b>

### Task 4

<b>Your query as a text</b>

< put here the screenshot of the output>

<b>Your comments (optional)</b>

### Task 5

<b>Your query as a text</b>

< put here the screenshot of the output>

<b>Your comments (optional)</b>

## Task 6

<b>Your query as a text</b>

< put here the screenshot of the output>

<b>Your comments (optional)</b>