Министерство науки и высшего образования РоссийскойФедерации
федеральное государственное автономное образовательноеучреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«ЗАПРОСЫ НА ВЫБОРКУ И МОДИФИКАЦИЮ ДАННЫХ, ПРЕДСТАВЛЕНИЯ И ИНДЕКСЫ В POSTGRESQL»

Студент: Злотникова Карина Александровна

Факультет: ИКТ

Группа: K32392

Преподаватель: Говорова М.М.

Дата: 02.05.2023

Санкт-Петербург 2023

Лабораторная №3

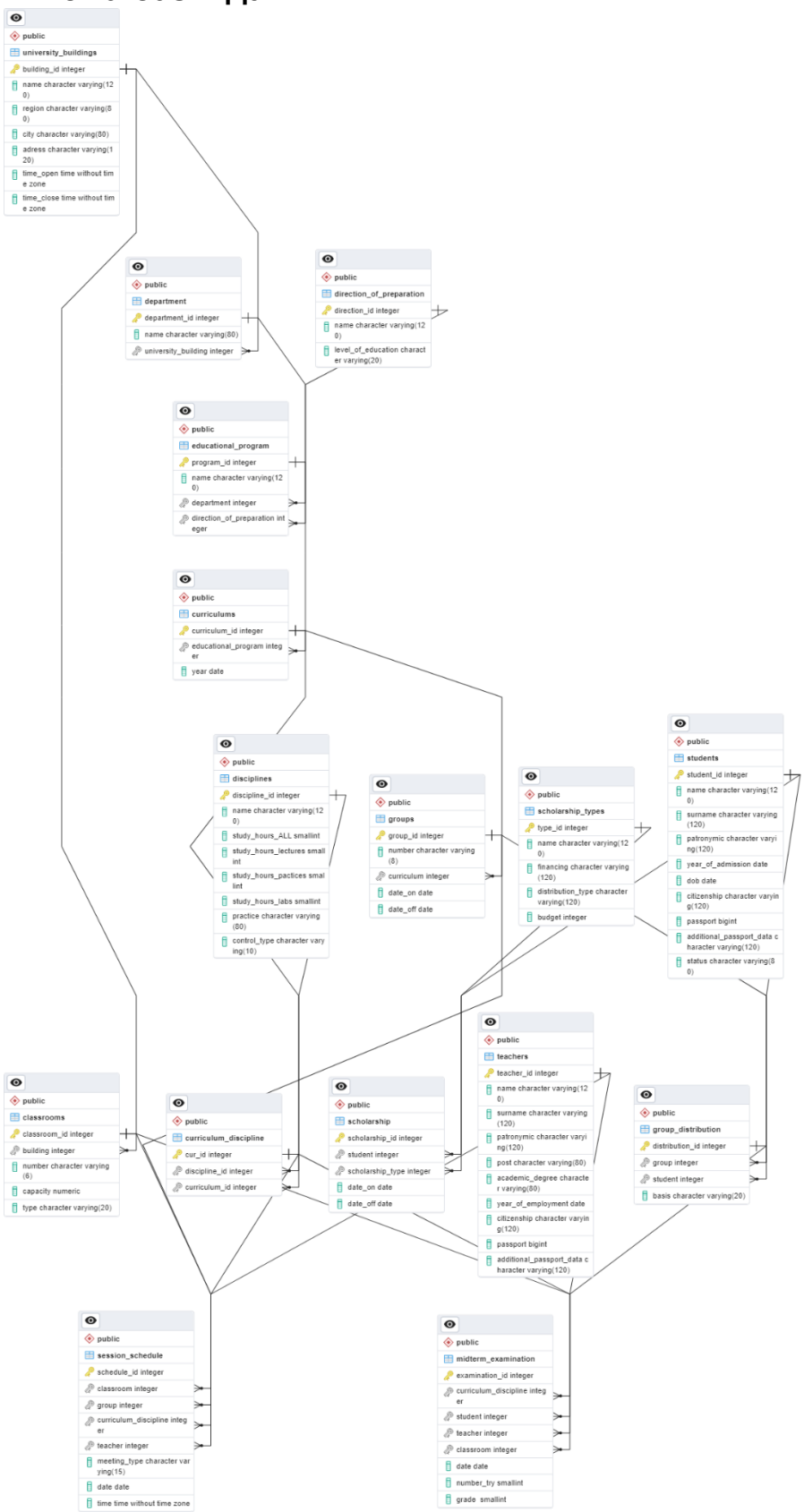
Процедуры:

- Для повышения стипендии отличникам на 10%.
- Для перевода студентов на следующий курс.
- Для изменения оценки при успешной пересдаче экзамена.

Триггеры:

- список студентов, получивших двойки на первой попытке с указанием фамилии преподавателя, которым они должны пересдать экзамен;
- данных о студентах при получении ими хотя бы одной оценки 2 (после 3-й попытки).

Схема базы данных:

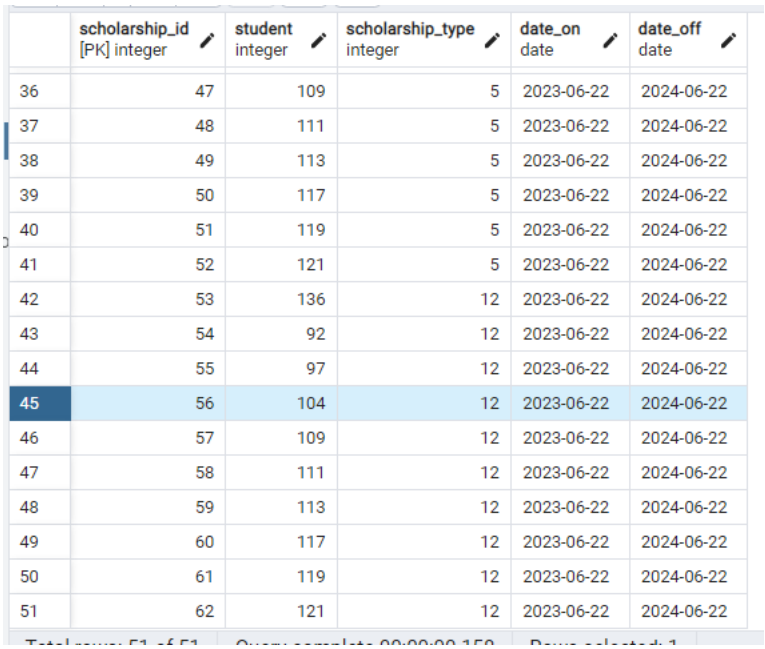


Процедуры:

- **Повышение стипендии отличникам на 10%.**

```
CREATE OR REPLACE PROCEDURE assign_scholarship_for_excellent_students()
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO scholarship (student, scholarship_type, date_on,
date_off)
 SELECT s.student_id, 12, current_date, (current_date + INTERVAL '1
year')
    FROM students s
    WHERE NOT EXISTS (
        SELECT 1
        FROM midterm_examination m
        WHERE m.student = s.student_id AND m.grade < 85
    );
END;
$$;
```

	scholarship_id [PK] integer	student integer	scholarship_type integer	date_on date	date_off date
26	27	109	12	2023-06-22	2024-06-22
27	28	111	12	2023-06-22	2024-06-22
28	29	113	12	2023-06-22	2024-06-22
29	30	117	12	2023-06-22	2024-06-22
30	31	119	12	2023-06-22	2024-06-22
31	32	121	12	2023-06-22	2024-06-22
32	43	136	5	2023-06-22	2024-06-22
33	44	92	5	2023-06-22	2024-06-22
34	45	97	5	2023-06-22	2024-06-22
35	46	104	5	2023-06-22	2024-06-22
36	47	109	5	2023-06-22	2024-06-22
37	48	111	5	2023-06-22	2024-06-22
38	49	113	5	2023-06-22	2024-06-22
39	50	117	5	2023-06-22	2024-06-22
40	51	119	5	2023-06-22	2024-06-22
41	52	121	5	2023-06-22	2024-06-22



	scholarship_id [PK] integer	student integer	scholarship_type integer	date_on date	date_off date
36	47	109	5	2023-06-22	2024-06-22
37	48	111	5	2023-06-22	2024-06-22
38	49	113	5	2023-06-22	2024-06-22
39	50	117	5	2023-06-22	2024-06-22
40	51	119	5	2023-06-22	2024-06-22
41	52	121	5	2023-06-22	2024-06-22
42	53	136	12	2023-06-22	2024-06-22
43	54	92	12	2023-06-22	2024-06-22
44	55	97	12	2023-06-22	2024-06-22
45	56	104	12	2023-06-22	2024-06-22
46	57	109	12	2023-06-22	2024-06-22
47	58	111	12	2023-06-22	2024-06-22
48	59	113	12	2023-06-22	2024-06-22
49	60	117	12	2023-06-22	2024-06-22
50	61	119	12	2023-06-22	2024-06-22
51	62	121	12	2023-06-22	2024-06-22

- **Перевод студентов на следующий курс.**

```
CREATE OR REPLACE PROCEDURE promote_students() AS $$
DECLARE
    old_curriculum_id INTEGER;
    new_curriculum_id INTEGER;
    new_group_id INTEGER;
    new_group_number VARCHAR(8);
    new_group_date_on DATE;
    new_group_date_off DATE;

BEGIN
    -- Получаем ID текущего учебного плана
    SELECT curriculum_id INTO old_curriculum_id
    FROM curriculums
    WHERE year < CURRENT_DATE - INTERVAL '1 year';

    -- Если не найден предыдущий учебный план, игнорируем и выходим из
процедуры
    IF old_curriculum_id IS NULL THEN
        RETURN;
    END IF;

    -- Создаем новую группу только если найден новый учебный план
    SELECT curriculum_id INTO new_curriculum_id
    FROM curriculums
    WHERE year > CURRENT_DATE - INTERVAL '4 year';

    -- Если не найден новый учебный план, игнорируем и выходим из
процедуры
    IF new_curriculum_id IS NULL THEN
        RETURN;
    END IF;

    -- Генерируем номер новой группы
    SELECT MAX(group_id) + 1 INTO new_group_id
    FROM groups;
```

```
    new_group_number := 'Н' || new_group_id;

    -- Устанавливаем даты начала и окончания новой группы
    new_group_date_on := CURRENT_DATE;
    new_group_date_off := (CURRENT_DATE + INTERVAL '1 year');

    -- Вставляем новую группу в таблицу groups
    INSERT INTO groups (group_id, number, curriculum, date_on, date_off)
    OVERRIDING SYSTEM VALUE
 VALUES (new_group_id, new_group_number, new_curriculum_id,
new_group_date_on, new_group_date_off);

    -- Распределяем студентов в новую группу
    INSERT INTO group_distribution ("group", student, basis)
 SELECT new_group_id, student_id, (
        SELECT basis
        FROM group_distribution
        WHERE student = student_id
        LIMIT 1
 )
 FROM students
 WHERE status = 'студент';

    -- Коммитим изменения
    COMMIT;
END;
$$ LANGUAGE plpgsql;
```

	name character varying (120)	surname character varying (120)	group_number character varying (8)
1	Иван	Иванов	ИУ522
2	Петр	Петров	ИУ522
3	Анна	Сидорова	ИУ522
4	Максим	Козлов	ИУ522
5	Дарья	Михайлова	ИУ522
6	Сергей	Николаев	ИУ522
7	Алиса	Ковалева	ИУ522
8	Иван	Иванов	ИУ522
9	Иван	Иванов	ИУ522
10	Иван	Иванов	ИУ522
11	Иван	Иванов	ИУ522
12	Алексей	Лебедев	ПМ420
13	Виктория	Егорова	ИУ721
14	Георгий	Федоров	БИ123
15	Иван	Иванов	ИУ522

;

	character varying (120)	character varying (120)	character varying (8)
14	Георгий	Федоров	БИ123
15	Иван	Иванов	Н13
16	Петр	Петров	Н13
17	Анна	Сидорова	Н13
18	Максим	Козлов	Н13
19	Дарья	Михайлова	Н13
20	Сергей	Николаев	Н13
21	Алиса	Ковалева	Н13
22	Георгий	Федоров	Н13
23	Алексей	Лебедев	Н13
24	Виктория	Егорова	Н13
25	Иван	Козлов	Н13
26	Анна	Новикова	Н13
27	Михаил	Воробьев	Н13
28	Александр	Калинин	Н13
29	Виктория	Максимова	Н13

- **Изменение оценки при успешной пересдаче экзамена.**

```
CREATE OR REPLACE PROCEDURE update_grade(
    student_id_temp INTEGER,
    exam_id INTEGER,
    new_grade INTEGER
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Проверяем, что студент существует
    IF NOT EXISTS (SELECT 1 FROM students WHERE student_id =
student_id_temp) THEN
        RAISE EXCEPTION 'Студент с идентификатором % не найден.',
student_id_temp;
    END IF;

    -- Проверяем, что экзамен существует
    IF NOT EXISTS (SELECT 1 FROM midterm_examination WHERE examination_id
= exam_id) THEN
        RAISE EXCEPTION 'Экзамен с идентификатором % не найден.',
exam_id;
    END IF;

    -- Проверяем, что студент сдал экзамен
    IF EXISTS (SELECT 1 FROM midterm_examination WHERE student =
student_id_temp AND examination_id = exam_id) THEN
        -- Обновляем оценку
        UPDATE midterm_examination
        SET grade = new_grade
        WHERE student = student_id_temp AND examination_id = exam_id;

        -- Выводим сообщение об успешном изменении оценки
        RAISE NOTICE 'Оценка для студента с идентификатором % по экзамену
с идентификатором % успешно изменена на %.', student_id_temp, exam_id,
new_grade;
```

```
    ELSE
        RAISE EXCEPTION 'Студент с идентификатором % не сдавал экзамен с
идентификатором %.', student_id_temp, exam_id;
    END IF;
END;
$$;
```

	examination_id [PK] integer	curriculum_discipline integer	student integer	teacher integer	classroom integer	date date	number_try smallint	grade smallint
14	3336	1	4	8	1	2023-05-20	1	54
15	3337	1	5	8	1	2023-05-20	1	83
16	3338	1	6	8	1	2023-05-20	1	84
17	3339	1	7	8	1	2023-05-20	1	57
18	3340	1	8	8	1	2023-05-20	1	99
19	3341	1	9	8	1	2023-05-20	1	76
20	3342	1	10	8	1	2023-05-20	1	93
21	3343	1	86	8	1	2023-05-20	1	92
22	3344	1	87	8	1	2023-05-20	1	79

CALL update_grade(5, 3337, 90);

```
ЗАМЕЧАНИЕ:  Оценка для студента с идентификатором 5 по экзамену с идентификатором 3337 успешно изменена на 90.
CALL
```

	examination_id [PK] integer	curriculum_discipline integer	student integer	teacher integer	classroom integer	date date	number_try smallint	grade smallint
1	3326	1	5	8	1	2023-05-20	1	50
2	3391	2	5	9	2	2023-05-21	1	88
3	3445	3	5	10	3	2023-05-22	1	94
4	3337	1	5	8	1	2023-05-20	1	90
5	3499	4	5	11	4	2023-05-23	1	87

Триггер:

- Создаём журнал записи:

```
CREATE TABLE public.log (
id SERIAL PRIMARY KEY,
operation VARCHAR(50) NOT NULL,
timestamp TIMESTAMP WITHOUT TIME ZONE DEFAULT (now() AT TIME ZONE 'utc'),
tablename VARCHAR(255),
old_data TEXT,
new_data TEXT
);
```

- Создаём функцию, которую триггеры будут вызывать и через которую будут записывать:

```
CREATE OR REPLACE FUNCTION log_changes() RETURNS TRIGGER AS $$
BEGIN
IF (TG_OP = 'DELETE') THEN
INSERT INTO public.log (operation, tablename, old_data) VALUES (TG_OP,
TG_TABLE_NAME, old::text);
RETURN old;
```

```
ELSIF (TG_OP = 'UPDATE') THEN
INSERT INTO public.log (operation, tablename, old_data, new_data) VALUES
(TG_OP, TG_TABLE_NAME, old::text, new::text);
RETURN new;
ELSIF (TG_OP = 'INSERT') THEN
INSERT INTO public.log (operation, tablename, new_data) VALUES (TG_OP,
TG_TABLE_NAME, new::text);
RETURN new;
END IF;
RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

- Создаём триггер для каждой таблицы БД:

```
CREATE TRIGGER log_classrooms_changes
AFTER INSERT OR UPDATE OR DELETE ON classrooms
FOR EACH ROW EXECUTE PROCEDURE log_changes();


CREATE TRIGGER log_curriculum_discipline_changes
AFTER INSERT OR UPDATE OR DELETE ON curriculum_discipline
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_curriculums_changes
AFTER INSERT OR UPDATE OR DELETE ON curriculums
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_department_changes
AFTER INSERT OR UPDATE OR DELETE ON department
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_direction_of_preparation_changes
AFTER INSERT OR UPDATE OR DELETE ON direction_of_preparation
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_disciplines_changes
AFTER INSERT OR UPDATE OR DELETE ON disciplines
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_educational_program_changes
AFTER INSERT OR UPDATE OR DELETE ON educational_program
FOR EACH ROW EXECUTE PROCEDURE log_changes();


CREATE TRIGGER log_group_distribution_changes
AFTER INSERT OR UPDATE OR DELETE ON group_distribution
FOR EACH ROW EXECUTE PROCEDURE log_changes();


CREATE TRIGGER log_groups_changes
AFTER INSERT OR UPDATE OR DELETE ON "groups"
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_midterm_examination_changes
AFTER INSERT OR UPDATE OR DELETE ON midterm_examination
FOR EACH ROW EXECUTE PROCEDURE log_changes();
```

```
CREATE TRIGGER log_scholarship_changes
AFTER INSERT OR UPDATE OR DELETE ON scholarship
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_scholarship_types_changes
AFTER INSERT OR UPDATE OR DELETE ON scholarship_types
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_session_schedule_changes
AFTER INSERT OR UPDATE OR DELETE ON session_schedule
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_students_changes
AFTER INSERT OR UPDATE OR DELETE ON students
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_teachers_changes
AFTER INSERT OR UPDATE OR DELETE ON teachers
FOR EACH ROW EXECUTE PROCEDURE log_changes();

CREATE TRIGGER log_university_buildings_changes
AFTER INSERT OR UPDATE OR DELETE ON university_buildings
FOR EACH ROW EXECUTE PROCEDURE log_changes();
```