ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«Казанский (Приволжский) Федеральный Университет»
Институт вычислительной математики и информационных технологий
Кафедра прикладной математики и искусственного интеллекта

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

курса «Базы данных»

Вариант 4

Выполнил студент:

Визель Леонид Александрович
группа: 09-122

Проверил:

к.н, доцент
Бухараев Наиль Раисович

Казань, 2023 г.

Содержание

Глава 1

Цель работы

- Ознакомится с синтаксисом SQL;
- Научится создавать таблицы и писать запросы
- Научится эффективно комментировать собственный код
- Протестировать всё выше перечисленное на практике

Глава 2

Задание

- Создать три таблицы.
- Описание таблиц включает использование (хотя бы по одному разу): NOT NULL, DEFAULT, PRIMARY KEY, CHECK и IDENTITY.
- Описаны две межтабличные связи:
- Одна без использования системного каскадного удаления и обновления. К этой связи определен триггер на каскадное удаление в дочерней таблице. Создан второй триггер на какое-либо событие.
- Другая с использованием системного каскадного удаления и обновления (ON DELETE CASCADE ON UPDATE CASCADE).
- Создана хранимая процедура с выходными параметрами
- Подготовлен SQL-script для загрузки данных в таблицы и данные загружены (не менее пяти строк в каждой таблице). База данных корректна в смысле явно описанных ограничений целостности.

Глава 3

Основная часть

3.1. Подготовка

Для ознакомления с синтаксисом языка необходимо обратиться на основной сайт PostgreSQL - https://www.postgresql.org/docs/16/index.html.

Для загрузки программного обеспечения использовалась ссылка - https://www.enterprisedb.com/downloads/postgres-postgresql-downloads и последняя стабильная версия на момент декабря 2023 года - 16.1

Для тестирования запросов использовалось программное обеспечение pgAdmin 4 v8.0, поставляемое как часть пакета из вышеописанной ссылки.

Полностью расписанное задание находится по адресу https://github.com/Leonid-Vizel/DataBaseHomeWork и является публичным ресурсом.

3.2. Листинг скрипта создания таблиц

Для создания таблиц используем команду CREATE TABLE, к которой добавим приписку IF NOT EXISTS для упрощения тестирования.

Аналогичное применим для индексов.

К сожалению создание процедур не поддерживает синтаксис IF NOT EXISTS, потому придётся обойтись без этого.

По заданию необходимо построить 3 таблицы: (5) Пооперационные нормы затрат труда на изготовление, (6) Тарифный справочник, (7а) Справочник профессий рабочих.

Однако для эффективного построениия запросов по указанным заданиям необходимо достроить ещё 3 таблицы: материалы, детали, операции.

Листинг 3.1: Скрипт создания таблиц

```
-- Создаём таблицу "Materials" - материалы
-- Первичному ключу даём название "Id" и говорим, что он должен генерироваться самостоятельно базой данных (+ дописываем CONSTRAINT "PK_Materials" PRIMARY KEY ("Id") соблюдая конфенцию о именовании CONSTRAINT описывающих первичные ключи)
-- Даём имя материалу - "Name", это обязательная информация, так что указываем NOT NULL, а также чтобы не ограничивать пользователей используем character varying(1000), пусть лучше база поднапряжётся, но наши запросы будут выглоядет чище, а пользователи будут довольны
-- Столбец "Measurement" - описывает единицы изменения материала, он вряд ли будет очень большим, выделим 100 знаков под него (character varying(100)), эта информация обязательна (NOT NULL)
-- "PlanPrice" - плановая цена на материал, должна быть числом с плавающей запятой, даём пользователю максимальную свободу - используем numberic до(131072 знаков до запятой, и 16383 после), эта информация обязательна (NOT NULL)
CREATE TABLE IF NOT EXISTS "Materials" (
    "Id" integer GENERATED BY DEFAULT AS IDENTITY,
    "Name" character varying(1000) NOT NULL,
    "Measurement" character varying(100) NOT NULL,
    "PlanPrice" numeric NOT NULL,
    CHECK ("PlanPrice" > 0),
    CONSTRAINT "PK_Materials" PRIMARY KEY ("Id")
);

-- По аналогии с первой таблицей строим таблицы соблюдая конвенции и стараясь максимально угодить пользователю

-- Таблица деталей
-- Id - автоматически генерируемый первичный ключ
-- DetailType - Тип детали обязательный(параметр)
-- Name - Название детали обязательный(параметр) максимальный(размер - 2000 символов)
-- Measurement - Единицы измерения детали обязательный(параметр)
```

```
    максимальный( размер – 500 символов) шт('' например)
22 -- PlanPrice – Плановая стоимость детали обязательный( параметр)
23 CREATE TABLE IF NOT EXISTS "Parts" (
24     "Id" integer GENERATED BY DEFAULT AS IDENTITY,
25     "DetailType" integer NOT NULL,
26     "Name" character varying(2000) NOT NULL,
27     "Measurement" character varying(500) NOT NULL,
28     "PlanPrice" numeric NOT NULL,
29     CONSTRAINT "PK_Parts" PRIMARY KEY ("Id")
30 );
31
32 -- Таблица профессий
33 -- Id – автоматически генерируемый первичный ключ
34 -- Name – Название профессии обязательный( параметр) максимальный( размер –
    1000 символов)
35 CREATE TABLE IF NOT EXISTS "Professions" (
36     "Id" integer GENERATED BY DEFAULT AS IDENTITY,
37     "Name" character varying(1000) NOT NULL,
38     CONSTRAINT "PK_Professions" PRIMARY KEY ("Id")
39 );
40
41 -- Таблица профессий
42 -- Id – автоматически генерируемый первичный ключ
43 -- PerHour – Тарифная ставка обязательный( параметр)
44 CREATE TABLE IF NOT EXISTS "Tarifs" (
45     "Id" integer GENERATED BY DEFAULT AS IDENTITY,
46     "PerHour" numeric NOT NULL,
47     CONSTRAINT "PK_Tarifs" PRIMARY KEY ("Id")
48 );
49
50 -- В отличии от прошлых таблиц следующие имеют внешние ключи, следуя
    конвенции начинаем названия CONSTRAINT с FK, а также применяем ON DELETE
    CASCADE для автоматического каскадного() удаления строк из базы при удалении
    родительских сущностей
51
52 -- Таблица операций
53 -- Id – автоматически генерируемый первичный ключ
54 -- PartId – Обязательный внешний ключ с каскадным удалением ссылка( на
    таблицу деталей) ОДИН( КО МНОГИМ относительно таблицы деталей, то есть
    на одну деталь много операций)
55 -- MaterialId – Обязательный внешний ключ с каскадным удалением ссылка( на
    таблицу материалов) ОДИН( КО МНОГИМ относительно таблицы материалов, то
    есть на один материал много операций)
56 -- NeededAmount – Необходимое количество материала обязательное()
57 CREATE TABLE IF NOT EXISTS "Operations" (
58     "Id" integer GENERATED BY DEFAULT AS IDENTITY,
59     "PartId" integer NOT NULL,
60     "MaterialId" integer NOT NULL,
61     "NeededAmount" numeric NOT NULL,
62     CONSTRAINT "PK_Operations" PRIMARY KEY ("Id"),
```

```
63      CONSTRAINT "FK_Operations_Materials_MaterialId" FOREIGN KEY ("
        MaterialId") REFERENCES "Materials" ("Id") ON DELETE CASCADE,
64      CONSTRAINT "FK_Operations_Parts_PartId" FOREIGN KEY ("PartId")
        REFERENCES "Parts" ("Id") ON DELETE CASCADE
65  );
66
67  -- Здесь используем составной ключ, потому столбец "Id" не нужен
68
69  -- Таблица пооперационных нормы затрат труда на изготовление
70  -- PartId - Обязательный внешний ключ с каскадным обновлением ссылка( на
        таблицу деталей)
71  -- Не до конца ясно зачем нужен PartId, так как его мы можем достать и из
        таблицы операций. Но продолжаем идти по ТЗ
72  -- OperationId - Обязательный внешний ключ с каскадным удалением ссылка( на
        таблицу операций)
73  -- ProfessionId - Обязательный внешний ключ с каскадным удалением ссылка( на
        таблицу профессий)
74  -- TarifId - Обязательный внешний ключ с каскадным удалением ссылка( на
        таблицу тарифов)
75  -- Qualification - квалификационный разряд рабочего обязательное( текстовое
        поле с максимальным размеров в 2000 символов)
76  -- FinalMinuteTime - время подготовительнозаключительное- в( мин.);
77  -- PieceMinuteTime - время штучное в( мин.).
78  -- Не забываем  описать каскадный ключ состоящий из 4 внешних: PartId,
        OperationId, ProfessionId, TarifId
79  CREATE TABLE IF NOT EXISTS "Norms" (
80      "PartId" integer NOT NULL,
81      "OperationId" integer NOT NULL,
82      "ProfessionId" integer NOT NULL,
83      "TarifId" integer NOT NULL,
84      "Qualification" character varying(2000) NOT NULL,
85      "FinalMinuteTime" numeric NOT NULL,
86      "PieceMinuteTime" numeric NOT NULL,
87      CONSTRAINT "PK_Norms" PRIMARY KEY ("PartId", "OperationId", "
        ProfessionId", "TarifId"),
88      CONSTRAINT "FK_Norms_Operations_OperationId" FOREIGN KEY ("
        OperationId") REFERENCES "Operations" ("Id") ON UPDATE CASCADE,
89      CONSTRAINT "FK_Norms_Parts_PartId" FOREIGN KEY ("PartId")
        REFERENCES "Parts" ("Id") ON DELETE CASCADE,
90      CONSTRAINT "FK_Norms_Professions_ProfessionId" FOREIGN KEY ("
        ProfessionId") REFERENCES "Professions" ("Id") ON DELETE CASCADE,
91      CONSTRAINT "FK_Norms_Tarifs_TarifId" FOREIGN KEY ("TarifId")
        REFERENCES "Tarifs" ("Id") ON DELETE CASCADE
92  );
93
94  -- Создаём индексы, продолжая придерживаться конвенций начинаем( с IX).
        Индексы создаём только для самых вероятных использваний - внешних ключей
95
96  CREATE INDEX IF NOT EXISTS "IX_Norms_OperationId" ON "Norms" ("
        OperationId");
```

```
97 CREATE INDEX IF NOT EXISTS "IX_Norms_ProfessionId" ON "Norms" ("
       ProfessionId");
98 CREATE INDEX IF NOT EXISTS "IX_Norms_TarifId" ON "Norms" ("TarifId");
99 CREATE INDEX IF NOT EXISTS "IX_Norms_PartId" ON "Norms" ("PartId");
100 CREATE INDEX IF NOT EXISTS "IX_Operations_MaterialId" ON "Operations"
       ("MaterialId");
101 CREATE INDEX IF NOT EXISTS "IX_Operations_PartId" ON "Operations" ("
       PartId");
102
103 -- Создаём процедуру для частичного наполнения БД
104 -- Задаём ей имя Init
105 -- Обозначем, что пишем на SQL Странно( если бы здесь писали на чёмто- ином)
106 -- Между AS $$ и $$ вписываем комманды, которые хотим выполнить, в нашем
       случае - INSERT
107
108 CREATE PROCEDURE Init()
109 LANGUAGE SQL
110 AS $$
111 INSERT INTO "Materials"("Name", "Measurement", "PlanPrice") VALUES('
       Тестовый материал 1','кг',11.0);
112 INSERT INTO "Materials"("Name", "Measurement", "PlanPrice") VALUES('
       Тестовый материал 2','кг',12.0);
113 INSERT INTO "Materials"("Name", "Measurement", "PlanPrice") VALUES('
       Тестовый материал 3','кг',13.0);
114 INSERT INTO "Materials"("Name", "Measurement", "PlanPrice") VALUES('
       Тестовый материал 4','кг',14.0);
115 INSERT INTO "Materials"("Name", "Measurement", "PlanPrice") VALUES('
       Тестовый материал 5','кг',15.0);
116 INSERT INTO "Materials"("Name", "Measurement", "PlanPrice") VALUES('
       Тестовый материал 6','кг',16.0);
117
118 INSERT INTO "Professions"("Name") VALUES('Тестовый кадр 1');
119 INSERT INTO "Professions"("Name") VALUES('Тестовый кадр 2');
120 INSERT INTO "Professions"("Name") VALUES('Тестовый кадр 3');
121 INSERT INTO "Professions"("Name") VALUES('Тестовый кадр 4');
122 INSERT INTO "Professions"("Name") VALUES('Тестовый кадр 5');
123 INSERT INTO "Professions"("Name") VALUES('Тестовый кадр 6');
124
125 INSERT INTO "Tarifs"("PerHour") VALUES(100);
126 INSERT INTO "Tarifs"("PerHour") VALUES(101);
127 INSERT INTO "Tarifs"("PerHour") VALUES(102);
128 INSERT INTO "Tarifs"("PerHour") VALUES(103);
129 INSERT INTO "Tarifs"("PerHour") VALUES(104);
130 INSERT INTO "Tarifs"("PerHour") VALUES(105);
131
132 INSERT INTO "Parts"("Name", "DetailType", "Measurement", "PlanPrice")
       VALUES('Тестовая деталь 1', 0, 'кг', 12.7);
133 INSERT INTO "Parts"("Name", "DetailType", "Measurement", "PlanPrice")
       VALUES('Тестовая деталь 2', 1, 'кг', 100.9);
134 INSERT INTO "Parts"("Name", "DetailType", "Measurement", "PlanPrice")
```

```
    VALUES('Тестовая деталь 3', 0, 'кг', 17564);
135 INSERT INTO "Parts"("Name", "DetailType", "Measurement", "PlanPrice")
    VALUES('Тестовая деталь 4', 1, 'кг', 987.2);
136 INSERT INTO "Parts"("Name", "DetailType", "Measurement", "PlanPrice")
    VALUES('Тестовая деталь 5', 0, 'кг', 50.2);
137
138 INSERT INTO "Operations"("PartId", "MaterialId", "NeededAmount")
    VALUES(1, 1, 1);
139 INSERT INTO "Operations"("PartId", "MaterialId", "NeededAmount")
    VALUES(1, 2, 2);
140 INSERT INTO "Operations"("PartId", "MaterialId", "NeededAmount")
    VALUES(2, 3, 3);
141 INSERT INTO "Operations"("PartId", "MaterialId", "NeededAmount")
    VALUES(2, 4, 4);
142 INSERT INTO "Operations"("PartId", "MaterialId", "NeededAmount")
    VALUES(3, 5, 5);
143
144 INSERT INTO "Norms"("PartId", "OperationId", "ProfessionId", "TarifId"
    , "Qualification", "FinalMinuteTime", "PieceMinuteTime") VALUES(1,
    1, 1, 1, 'Тестовая квалификация 1', 1, 1);
145 INSERT INTO "Norms"("PartId", "OperationId", "ProfessionId", "TarifId"
    , "Qualification", "FinalMinuteTime", "PieceMinuteTime") VALUES(1,
    2, 2, 2, 'Тестовая квалификация 2', 2, 2);
146 INSERT INTO "Norms"("PartId", "OperationId", "ProfessionId", "TarifId"
    , "Qualification", "FinalMinuteTime", "PieceMinuteTime") VALUES(2,
    3, 3, 3, 'Тестовая квалификация 3', 3, 3);
147 INSERT INTO "Norms"("PartId", "OperationId", "ProfessionId", "TarifId"
    , "Qualification", "FinalMinuteTime", "PieceMinuteTime") VALUES(2,
    4, 4, 4, 'Тестовая квалификация 4', 4, 4);
148 INSERT INTO "Norms"("PartId", "OperationId", "ProfessionId", "TarifId"
    , "Qualification", "FinalMinuteTime", "PieceMinuteTime") VALUES(3,
    5, 5, 5, 'Тестовая квалификация 5', 5, 5);
149 $$;
150
151 -- Создаём простенькую функцию с возвращаемыми значениями.
152
153 CREATE OR REPLACE FUNCTION return_setof_int() RETURNS SETOF int AS
154 $$
155 BEGIN
156   RETURN NEXT 1;
157   RETURN NEXT 2;
158   RETURN NEXT 3;
159 END
160 $$ LANGUAGE plpgsql;
```

3.3. Листинг скрипта первого запроса

Для всех запросов используем команду SELECT. Для обращения по внешним ключам будем использовать INNER JOIN, так как он гарантирует наличие и родительской и дочерней сущностей.

Также в выражении SELECT оставляем лишь необходимые параметры.

Задание 1: Сведения о затратах труда для заданной профессии: код детали; номер операции; код профессии; часовая тарифная ставка.

Допустим, что идентификатор заданной профессии равен 1.

Листинг 3.2: Скрипт запроса 1

```
1  -- ========
2  -- Задание #1
3  -- Сведения о затратах труда для заданной профессии: код детали; номер операции;
       код профессии; часовая тарифная ставка.
4  -- ========
5  -- 1) Обращаюсь к таблице "Norms" - Нормы затрат труда и беру из неё нужные
       параметры
6  -- 2) Через INNER JOIN то( есть гарантирую, что и строка из Norms и строка из
       Tarifs при декартовом произведении присутствуют)
7  -- по ключу TarifId таблицы Norms подулючаю другую таблцу к запросу
8  -- 3) Применяю условие
9  -- 4) Довольствуюсь результатом
10 SELECT n."PartId", n."OperationId", n."ProfessionId", t."PerHour" AS "
       TarifPerHour"
11 FROM "Norms" AS n
12 INNER JOIN "Tarifs" AS t ON n."TarifId" = t."Id"
13 WHERE n."ProfessionId" = 1;
```

3.4. Листинг скрипта второго запроса

Задание 2: Для каждой детали: код детали; стоимость подготовительно-заключительных работ; стоимость на штуку (суммарно по всем операциям).

Для написания второго запроса потребуется применить группировку, а также агрегатные функции для обработки потроенных групп.

Для группировки используем GROUP BY, а для получения финальных результатов применим агрегатную функцию SUM для суммирования результатов знапроса, а также функцию ROUND для округления результата.

Листинг 3.3: Скрипт запроса 2

```
-- ========
-- Задание #2
-- Для каждой детали: код детали; стоимость подготовительнозаключительных работ; стоимость на штуку суммарно( по всем операциям).
-- ========
-- 1) Подключаемся к таблице "Parts" и берём из неё код детали
-- 2) К ней подключаем таблицы "Tarifs" и "Operations" по внешнеим ключам "TarifId" и "OperationId" из "Norms" соответственно
-- 3) К ней подключаем таблицу "Operations" по внешнему ключу "OperationId" из "Norms"
-- 4) К ней подключаем таблицу "Materials" по внешнему ключу "MaterialId" из "Operations"
-- 6) Группируем данные для каждой детали
-- 5) В SELECT расчитываем округлённые до 1 знака суммы FinalWorkPrice стоимость( подготовительнозаключительных работ) не( забывая учесть перевод из минут в часы)
-- и MaterialPrice стоимость( на штуку)

SELECT p."Id",
ROUND(SUM(n."FinalMinuteTime" / 60.0 * t."PerHour"), 1) as "FinalWorkPrice",
ROUND(SUM(m."PlanPrice" * o."NeededAmount"), 1) as "MaterialPrice"
FROM "Parts" AS p
INNER JOIN "Norms" AS n ON n."PartId" = p."Id"
INNER JOIN "Tarifs" AS t ON n."TarifId" = t."Id"
INNER JOIN "Operations" AS o ON n."OperationId" = o."Id"
INNER JOIN "Materials" AS m ON o."MaterialId" = m."Id"
GROUP BY p."Id";
```

3.5. Листинг скрипта третьего запроса

Задание 3: Все профессии, такие что: для каждого кода работ (из 06) имеется деталь в сборке которой рабочий такой профессии выполняет такую работу.

Для третего запроса понадобится применить вложенные подзапросы. Для них буду использовать NOT EXISTS, внутрь которого помещу необходимый подзапрос.

Листинг 3.4: Скрипт запроса 3

```
-- ======
-- Задача 3
-- Все профессии, такие что: для каждого кода работ из(06) имеется деталь в сборке которой рабочий такой профессии выполняет такую работу.
-- ======
-- 1) Для начала перефразируем задачу, в итоге получим более читабельное' в реляционном плане условие':
-- Получить все профессии для которых не существует такого тарифного плана для которого не существует нормы
-- То(есть будут выданы профессии на которые имеются нормы по вмсем тарифам)
-- 2) Подключаемся к таблице Professions и берём оттуда нужные поля: "Id" и "Name"
-- 3) Применяем к ней условие EXISTS точнее(его версию с NOT, чтобы обратить булевый результат)
-- 4) Внутри NOT EXISTS составляем подзапрос, в котором будем проверять отсутствие строк:
-- 4.1) Обращаемся к таблице Tarifs и берём из неё Id
-- 4.2) Снова применяем NOT EXISTS и подключаем финальную таблицу
-- 4.2.1) Обращаемся к таблице Norms и берём из неё Id
-- 4.2.2) Берём лишь нужные нам строки (n."ProfessionId" = p."Id" AND n."TarifId" = t."Id")
-- 4.2.3) Закрываем скобки вложенных условий и довольствуемся результатом
--
-- Задача выглядит сложно, но врот ещё одно перефразирование:
-- Находит такие данные о таких профессиях,
-- для которых не существует таких тарифов,
-- для которых не не существует норм.
SELECT p."Id", p."Name" FROM "Professions" as p
WHERE NOT EXISTS (
    SELECT "Id"
    FROM "Tarifs" as t
    WHERE NOT EXISTS (
        SELECT "Id"
        FROM "Norms" n
        WHERE n."ProfessionId" = p."Id" AND n."TarifId" = t."Id"
    )
);
```

Глава 4

Выводы

Как результат был изучен синтаксис PostgreSQL, на практике разобрана работа с его коммандами в соответствии с конвенциями наименования и применением эффективного комментирования кода.