

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
Высшего профессионального образования
«Казанский (Приволжский) Федеральный университет»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра прикладной математики и искусственного интеллекта

Направление: 01.03.04 — Прикладная математика

Профиль: прикладная математика

ОТЧЕТ

по дисциплине: «Базы данных»

10 вариант

Выполнил: студент 09-122 группы

Н.М. Макурин

Проверил: преподаватель

Н.Р. Бухараев

Казань, 2023

Содержание

1	ВВЕДЕНИЕ	3
1.1	Общая часть. Постановка задачи	3
2	ХОД РАБОТЫ	4
2.1	Луковая архитектура приложения	4
2.2	Подходы для создания БД	5
2.3	Создание проекта	5
2.4	Написание SELECT запросов	9
3	ВЫВОДЫ	11

1. ВВЕДЕНИЕ

1.1. Общая часть. Постановка задачи

Создать три таблицы. Описание таблиц включает использование (хотя бы по одному разу): NOT NULL, DEFAULT, PRIMARY KEY, CHECK и IDENTITY.

Описаны две межтабличные связи:

1) 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.

2) 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.

Таблицы следующего формата:

1) Профессии – Идентификатор, Наименование.

2) Нормы затрат труда – Идентификатор, Идентификатор профессии, Идентификатор операции, Разряд рабочего, Время подготовительно-заключительное (в мин.), Время штучное (в мин.).

3) Детали – Идентификатор, Тип детали, Наименование детали, Мера измерения, Цена.

Таким образом связи между таблицами: Для нормы затрат труда много деталей и много профессий. Связь "один ко многим".

Развернем архитектуру проекта с использованием Entity Framework (ORM), ЯП C sharp, разберем подходы современной разработки приложения с использованием PostgreSQL.

2. ХОД РАБОТЫ

2.1. Луковая архитектура приложения

В современной разработке используют разные архитектурные решения, например монолитные, N-Tier архитектуры, луковые(Onion) и другие. В качестве основной архитектуры проекта выберем луковые. Суть данного решения проста: в центре всего бизнес-логика или прикладная логика приложения. Поверх нее Persistence слой или по-другому слой базы данных, а также Application слой(приложение). Еще поверх Application и Persistence слоев лежит слой View (представление). View слой – это консоль, десктопное окно или клиент сайта и т.д. Эту архитектуру также иногда называют "чистой" поскольку она позволяет подменять слои на другие без особых затрат, тем самым меняя в корне логику всего приложения.

Самое важное, что зависимости слоев идут "из краев во внутрь". Если сделать по-другому зависимости, то нарушается принцип L подстановки Лисков (SOLID). Существуют много похожих схем чистой архитектуры, одну из них можно визульно оценить на следующей картинке:

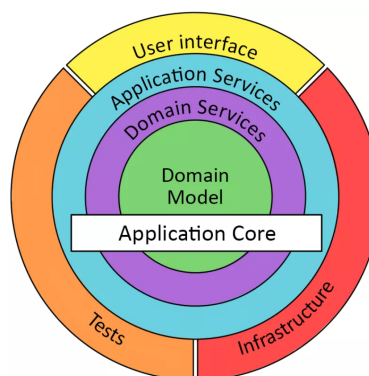


Рис. 1 – Одна из реализаций луковой архитектуры

2.2. Подходы для создания БД

В мире прикладного программирования выработались два основных подхода создания БД: Code First и Database first.

Code first подразумевает под себя создание бизнес-логики по принципам ООП, в классовой структуре и на их основе создаются таблицы (схемы) в БД. Для ускорения этого процесса были придуманы ORM (Object-Relational Mapping) системы, которые позволяют связать базы данных с классами из ООП с помощью маппинга. Мы не будем углубляться в подробности устройства ORM, но будем использовать этот способ, как основной способ создания базы данных. В качестве ORM используем Entity Framework, в качестве базы данных PostgreSQL.

Database First – подход, который требует создания схемы базы данных, а уже затем написания приложения под него. Вообще говоря, в PostgreSQL можно создавать таблицы ни разу не написав 'CREATE TABLE', при помощи внутреннего интерфейса.

2.3. Создание проекта

Создадим решение на C sharp, подключим Entity Framework, распишем в слое (проекте) Models, наши основные сущности : Standart, Detail, Profession, а также дополнительно сущности Tarif и Operation. Создадим миграцию для БД, пропишем строку подключения БД в файле appsettings.json. В нашем случае мы поднимаем сервер локально, поэтому используем стандартный порт для PostgreSQL, но при желании можно развернуть сервер и на хостинге, чтобы приложение было доступно в глобальном доступе. Таким образом структура приложения выглядит следующим образом:

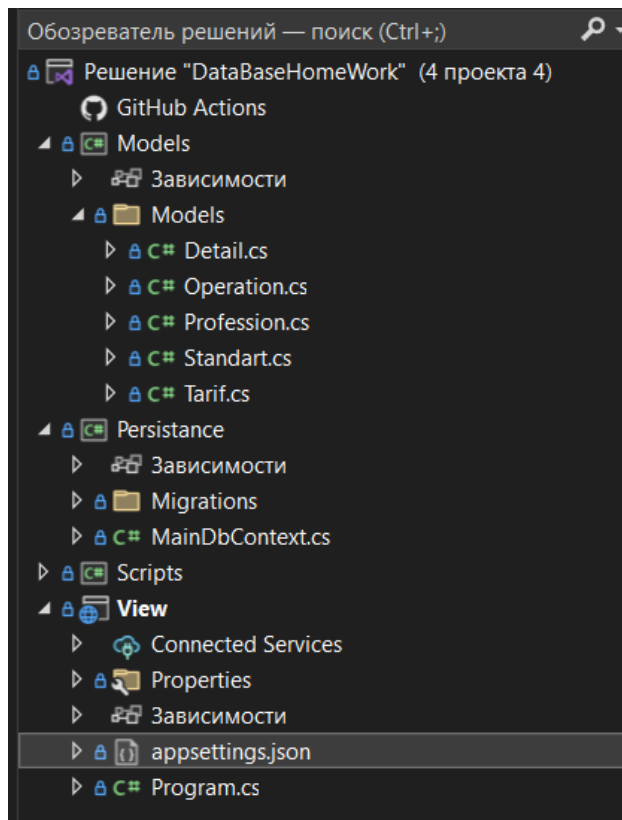


Рис. 2 – Структура приложения

После миграции ORM сгенерировала следующий скрипт:

```
CREATE TABLE "Details" (
    "Id" integer GENERATED BY DEFAULT AS IDENTITY,
    "TypeDetail" integer NOT NULL,
    "Name" character varying(1000) NULL,
    "Measurement" character varying(250) NULL,
    "Price" numeric NOT NULL,
    CONSTRAINT "FK_Details" PRIMARY KEY ("Id")
);

CREATE TABLE "Operations" (
    "Id" integer GENERATED BY DEFAULT AS IDENTITY,
    "Name" character varying(100) NOT NULL,
    CONSTRAINT "FK_Operations" PRIMARY KEY ("Id")
);

CREATE TABLE "Professions" (
    "Id" integer GENERATED BY DEFAULT AS IDENTITY,
    "Name" character varying(100) NOT NULL,
    CONSTRAINT "FK_Professions" PRIMARY KEY ("Id")
);

CREATE TABLE "Tarifs" (
    "Id" integer GENERATED BY DEFAULT AS IDENTITY,
    "CountPerHour" integer NOT NULL,
    "StandartId" integer NOT NULL,
    CONSTRAINT "FK_Tarifs" PRIMARY KEY ("Id")
);

CREATE TABLE "Standarts" (
    "Id" integer GENERATED BY DEFAULT AS IDENTITY,
    "OperationId" integer NOT NULL,
    "CodeProfession" integer NOT NULL,
    "DischargeBuilder" integer NOT NULL,
    "TarifId" integer NOT NULL,
    "TimeEnsure" time without time zone NOT NULL,
    "ValuePerTime" time without time zone NOT NULL,
    CONSTRAINT "FK_Standarts" PRIMARY KEY ("Id"),
    CONSTRAINT "DischargeBuilder" CHECK ("DischargeBuilder" > 0 AND "DischargeBuilder" < 9),
    CONSTRAINT "FK_Standarts_Operations_OperationId" FOREIGN KEY ("OperationId") REFERENCES "Operations" ("Id") ON DELETE CASCADE,
    CONSTRAINT "FK_Standarts_Tarifs_TarifId" FOREIGN KEY ("TarifId") REFERENCES "Tarifs" ("Id") ON DELETE CASCADE
);

CREATE INDEX "IX_Standarts_OperationId" ON "Standarts" ("OperationId");

CREATE UNIQUE INDEX "IX_Standarts_TarifId" ON "Standarts" ("TarifId");
```

Рис. 3 – Скрипт создания таблиц

Теперь перейдем в PostgreSQL и сделаем инициализацию данных (INSERT-scripts).

1) Для таблицы Professions:

```
INSERT INTO "Professions"("Id "Name") values(1,'Калибровщик')
```

```
INSERT INTO "Professions"("Id "Name") values(2,'Уборщик')
```

```
INSERT INTO "Professions"("Id "Name") values(3,'Директор')
```

```
INSERT INTO "Professions"("Id "Name") values(4,'Разнорабочий')
```

2) Для таблицы Details:

```
INSERT INTO "Details"("Id "Name "TypeDetail "Measurement "Price")  
values(0,'Коленвал ВАЗ',1,'КГ',1500);
```

```
INSERT INTO "Details"("Id "Name "TypeDetail "Measurement "Price")  
values(1,'Колесо ВАЗ',3,'КГ',100);
```

```
INSERT INTO "Details"("Id "Name "TypeDetail "Measurement "Price")  
values(2,'Рама ВАЗ',40,'КГ',40);
```

```
INSERT INTO "Details"("Id "Name "TypeDetail "Measurement "Price")  
values(3,'Двигатель ВАЗ',100,'КГ',450);
```

```
INSERT INTO "Details"("Id "Name "TypeDetail "Measurement "Price")  
values(4,'Ручник ВАЗ',200,'Г',25);
```

3) Для таблицы Standarts:

```
INSERT INTO "Standarts"  
("Id "OperationId "ProfessionId "DischargeBuilder  
"TarifId "TimeEnsure "ValuePerTime") values(0,2,2,8,4,'18:00','2:00',0);
```

```
INSERT INTO "Standarts"  
("Id "OperationId "ProfessionId "DischargeBuilder  
"TarifId "TimeEnsure "ValuePerTime") values(1,1,1,2,3,'20:30','5:00',3);
```

```
INSERT INTO "Standarts"  
("Id "OperationId "ProfessionId "DischargeBuilder  
"TarifId "TimeEnsure "ValuePerTime") values(2,1,2,1,1,'7:00','7:00',2);
```

```
INSERT INTO "Standarts"
```

```

("Id "OperationId "ProfessionId "DischargeBuilder
"TarifId "TimeEnsure "ValuePerTime") values(3,3,1,5,1,'18:30','2:40',1);
INSERT INTO "Standarts"
("Id "OperationId "ProfessionId "DischargeBuilder
"TarifId "TimeEnsure "ValuePerTime") values(4,2,4,4,0,'21:00','0:30',4);

```

Архитектура базы данных выглядит следующим образом:

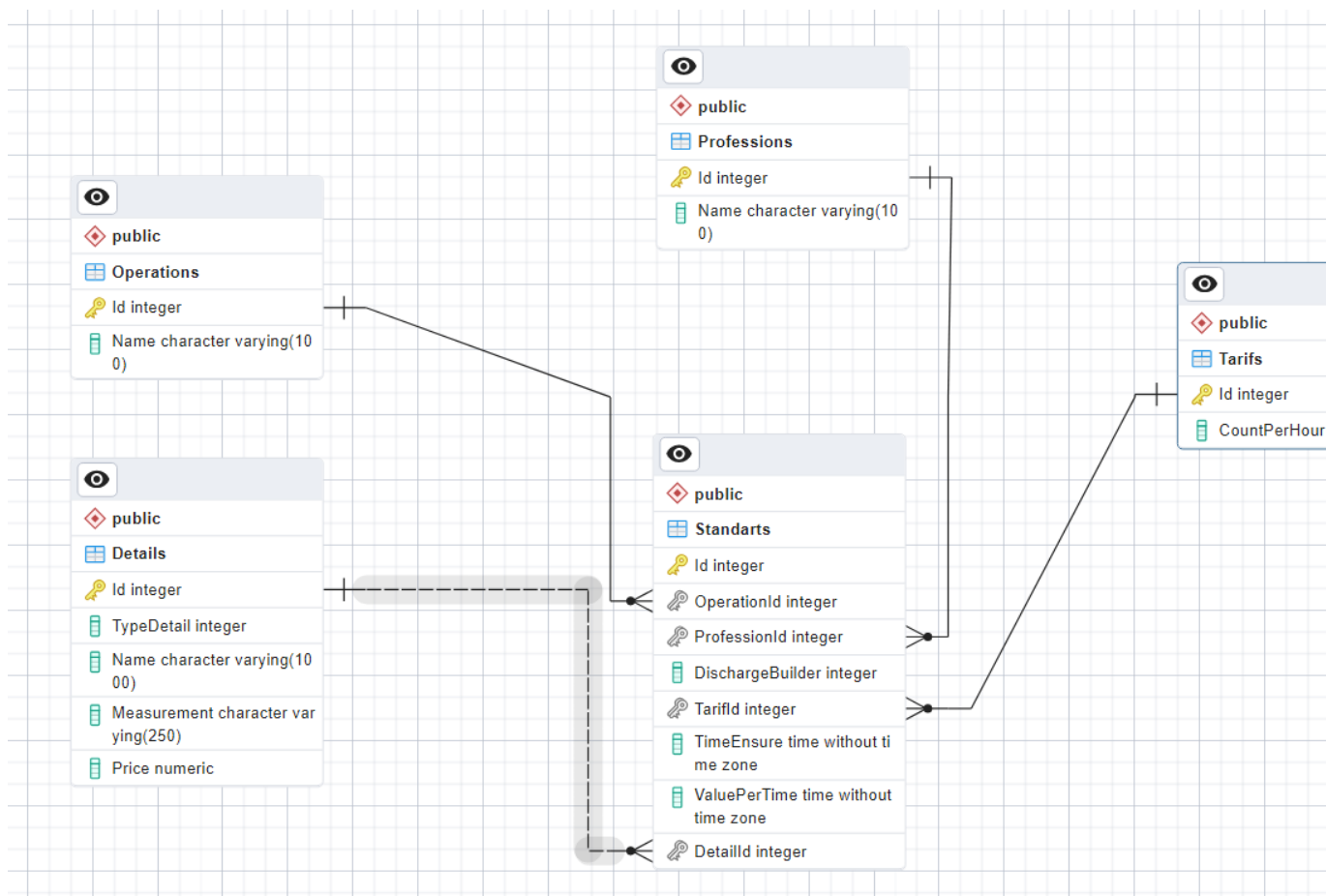


Рис. 4 – Архитектура БД

2.4. Написание SELECT запросов

Задача-1. Сведения о затратах труда в заданной операции: код детали; номер операции; наименование профессии; квалификационный разряд рабочего.

Для этого выберем, ту операцию из того стандарта в котором есть данная операция. Чтобы гарантировать что и строка из Standarts и строка из Profession при декартовом произведении присутствуют, использую INNER JOIN.

SELECT

```
Standart."DetailId",  
Standart."OperationId", Standart."ProfessionId",  
Operation."Name",  
Standart."DischargeBuilder"  
FROM "Standarts" AS Standart  
INNER JOIN "Operations" AS Operation ON Standart."OperationId"  
WHERE Standart."OperationId" = 1;
```

При текущей INSERT-конфигурации при выполнении запроса получится таблица:

	DetailId integer	OperationId integer	ProfessionId integer	Name character varying (100)	DischargeBuilder integer
1	3	1	1	Сработка	2
2	2	1	2	Сработка	1

Рис. 5 – Результат выполнения 1-ого запроса

Задача-2. Для каждой профессии: наименование; в сборке какого количества деталей рабочие этой профессии непосредственно участвуют.

Для этой задачи сгруппируем по атрибуту "Name" таблицу "Standarts" присоединенной с таблицей по внешнему ключу с таблицей

"Professions"и получим количество деталей через идентификатор детали.

```
SELECT "Name",COUNT(s." DetailId ")
FROM "Professions" as p
INNER JOIN "Standarts" as s ON s." ProfessionId" = p." Id"
GROUP BY "Name"
```

При текущей INSERT-конфигурации при выполнении запроса получится таблица:



	Name character varying (100) 	count bigint 
1	Директор	2
2	Разнорабочий	2
3	Уборщик	1

Рис. 6 – Результат выполнения 2-ого запроса

То есть директор и разнорабочий делает две детали, а уборщик только одну.

Задача-3. Все профессии, такие что: для каждой детали с ценой > 100 имеется операция по её сборке, которую должен выполнять рабочий этой профессии с разрядом > 4 .

Для этой задачи выберем наименование профессии и его идентификатор, такие что из таблицы "Details" атрибут "Price" > 100 и такие, что из таблицы "Standarts" вторичный ключ совпадает с ключом профессии и разряд рабочего больше 4. Также важно учесть, что деталь которую мы смотрели с ценой больше 100 совпадала с тем, что указана в таблице "Standarts". Для этого условия предлагаем следующий запрос:

```
SELECT p." Id ", p." Name" FROM "Professions" as p
WHERE EXISTS (
    SELECT " Price "
    FROM " Details " as d
```

```
WHERE d." Price">100 AND
EXISTS (
SELECT "Id"
FROM "Standarts" as s
WHERE s." ProfessionId" = p." Id"
AND s." DischargeBuilder">4
AND s." DetailId" = d." Id"
)
);
```

3. ВЫВОДЫ

Мы научились правильно проектировать приложение по "чистой архитектуре использовать ORM-системы для упрощения создания таблиц, писать SELECT запросы и подзапросы с применением различных команд SQL.