

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО
ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

***Федеральное государственное автономное
образовательное учреждение
высшего образования***

**«Национальный исследовательский ядерный университет
«МИФИ»»**

Кафедра №42 «Криптология и кибербезопасность»

ЛАБОРАТОРНАЯ РАБОТА №1-1:

«Построение модели данных»

Аверин Владислав

Группа Б19-505

Март, 2022

Содержание

Описание предметной области	3
Разработка	4
Нормализация.....	5
Листинг SQL.....	7
Выводы	9

Описание предметной области

В качестве легенды для рассматриваемой базы данных был выбран объект «Полицейский департамент». База данных, описывающая такую предметную область, должна содержать информацию как минимум обо всех своих служащих, делах (открытых и закрытых), а так же в достаточной мере описывать их взаимодействие друг с другом (какие дела какими сотрудниками расследуются/расследовались). При проектировании данной БД нужно учитывать несколько желательных характеристик:

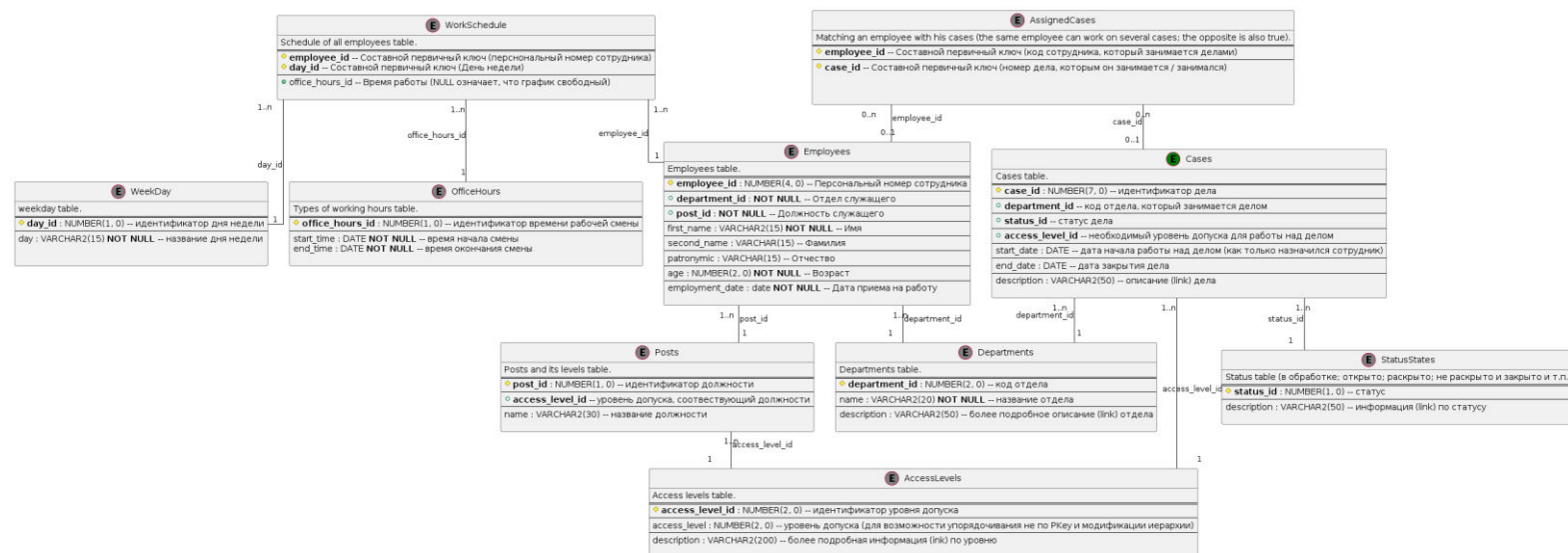
- Главная таблица фактов: ею разумно сделать таблицу кейсов (расследований), т.к. какие бы процессы в департаменте не происходили, все они взаимодействуют вокруг именно этой сущности; кроме того, полицейский департамент тем и отличается от других легенд, что его основная функция – расследовать различного рода преступления. Причем, дела, статус которых «закрыто»/«Не раскрыто» **не должны** удаляться из БД, т.к. основной ее функцией в данном контексте предметной области является накопление информации, т.е. она помимо прочего работает и как архив (что, кстати говоря, может быть проблемой производительности в будущем при большом кол-ве записей);
- Должны быть предусмотрены связи «Один сотрудник может работать над несколькими делами сразу» и «Одно дело могут расследовать одновременно несколько сотрудников». Это вполне логично, т.к. в реальности обычно именно такие связи. Причем, понятие «работать над делом» в моем понимании означает, что служащий хоть как-то с ним связан (например, выступает в роли судмедэксперта, что отражается в отчете);
- Нужно реализовать минимально необходимые в выбранной области атрибуты: уровень допуска (секретности) и классификацию как служащих, так и дел по этому атрибуту. Как вариант, можно было бы выдавать каждому сотруднику свой уровень допуска, однако я выбрал немного другой вариант: у каждого сотрудника есть своя должность, которая соотносится с уровнем допуска. И тот, и другой вариант имеет место быть, но второй мне почему-то нравится больше;
- Наличие разных отделов. Логично, что дело по ограблению банка не станет расследовать отдел по, например, наркоторговле, в полицейский департамент в моем понимании – это большая гос.структура, которая занимается огромным количеством расследований различного направления. Поэтому вполне разумно будет дать такие атрибуты и сущности сотрудников (который задается при приеме на работу или изменяется при переводе в другой отдел), и сущности дел (ибо опять же, ограблением банка не должен заниматься отдел по особо тяжким или криминалистике);

Разработка

Как было сказано выше, основным процессом в БД должна быть сущность дел. Помимо этого, какое-то учреждение подразумевает наличие персонала (сотрудников), поэтому еще одна сущность должна быть связана с работниками. Это два, можно сказать, основные свойства, а остальные таблицы будут вспомогательными таблицами измерений, для поддержания III нормальной формы (по сути, и таблицы сотрудников и дел тоже, т.к. у меня существует таблица AssignedCases сопоставления дел и сотрудников:

Изначально в легенде предполагалось, что одно дело могут вести несколько людей: в таком контексте в сущность Employees можно было бы добавить атрибут case_id, который мог бы быть NULL; однако как справедливо заметил наш руководитель по дисциплине, такое строение не позволяет без изменений БД организовать связь «один сотрудник может вести несколько дел». Поэтому я вынес обе эти связи в отдельную таблицу, где объявлены лишь два атрибута employee_id и case_id, являющиеся одновременно и составным первичным ключом, для исключения дубликатов.)

В результате нормализации до III формы я пришел к следующей таблице (она есть на git'e <https://github.com/Infernum/DataBase-Security/blob/main/ERD/ERD.png>):







В ней есть информация о:

- Расписании сотрудников (день недели + время работы);
- Всех делах, когда-либо заведенных;
- Сотрудниках со своими должностями и существующими отделами;
- Примитивная система управления доступом (на основе дискреционного управления доступом)

Нормализация


Докажем теперь, что предложенная схема базы данных нормализована до III формы:

1NF (соблюдение реляционных принципов)

Требование	Выполнение
Отсутствие дублирующих строк (по сути, наличие первичного ключа в каждой таблице)	
Каждый атрибут представляет собой атомарное (неделимое) значение	
Каждый атрибут хранит данные одного типа	
Отсутствие любого вида перечислений	

Таким образом, БД находится в первой нормальной форме.

2NF (условия первичного ключа)

Требование	Выполнение
БД находится в 1NF	
Каждая таблица имеет первичный ключ	
Каждый атрибут таблицы с составным основным ключом должен зависеть полностью от всего ключа	

Насчет последнего пункта: в моей схеме две таблицы с составным первичным ключом: AssignedCases (но так кроме самого ключа ничего нет, так что она не интересна) и WorkSchedule: в нем единственное поле office_hours_id зависит от обоих атрибутов ключа. Действительно, невозможно сказать, что время работы зависит только от сотрудника (и неважно, в какой день он работает) или от дня недели, а для всех сотрудников график одинаковый. То есть, мы не можем получить значение атрибута office_hours_id, зная только часть ключа. Таким образом, БД находится во II нормальной форме.

3NF (отсутствие транзитивной зависимости).

Для третьей нормальной форму необходимо сделать так, чтобы не было неключевых атрибутов, зависящих от других неключевых атрибутов. Если таковые имеются, необходимо как минимум создать еще одну таблицу измерений, куда вынести эти атрибуты отдельно.

- Для начала рассмотрим самые «маленькие» таблицы измерений:
 - Не имеют второстепенных атрибутов: **AssignedCases**;
 - 1 второстепенный атрибут: **WeekDay, WorkSchedule** (выступает в роли внешнего ключа, так что не считается), **StatusStates**.
 - 2 второстепенных атрибута: **OfficeHours** (все в порядке, start_time и end_time привязаны только к id режима работы), **Posts** (аналогично OfficeHours), **Departments** (можно сказать, что описание зависит от названия отдела, однако в конце концов description можно дать какой угодно), **AccessLevels** (та же история, что и с Departments; вообще говоря, эти таблицы и появились для отсутствия транзитивной зависимости в более высоких таблицах).
- Остаются таблицы Cases и Employees.
 - **Cases**. Start_date и end_date не могут в принципе ни от чего зависеть, кроме первичного ключа (идентификатора дела), это величина, можно сказать, случайная для каждого случая.

Здесь можно обсудить связь access_level_id с department_id. На самом деле это отдельная тема для рассуждений. Дело в том, что при проектировании легенды я полагал, что уровень допуска может быть у конкретной должности, а не у всего отдела в целом. То есть допуск определяется исключительно по должности. Значит, следовательно, независимо от того, в каком отделе трудится, будет иметь один и тот же уровень секретности. В принципе, как по мне, такой процесс имеет место быть (~~на самом деле, я просто прикинул, что будет, если попытаться создать связь уровня допуска с обоими атрибутами должности и отдела, понял, что будет геммор с таблицей дел и связями с другими таблицами и забил~~). Поэтому катастрофически важно создавая должности **НИ В КОЕМ СЛУЧАЕ** не допускать их корреляций с отделами: детектив есть детектив, да и вообще по-хорошему в моей легенде не будет каких-либо «особо секретных» дел (~~кроме одной пасхалки~~), поэтому это сделано больше для того, чтобы создать какую-никакую систему управления доступом, чтобы условные работники канцелярии не имели доступа к делам, а «самый важный дядька» имел полный доступ.

Можно было бы создать и другую зависимость: чтобы уровень допуска сотрудника определялся двумя показателями: его должностью и отделом, но это бы усложнило запросы и изменение данных: кроме того, что нужно было бы реорганизовать таблицу Employees, изменения бы коснулись и Cases, причем не самым лучшим образом: если уровень допуска зависит от должности и отдела, то как определять уровень допуска у дела, если мы знаем только зависимость допуска от двух параметров? И определять ли у него тогда department_id? И как соотносить работника с делом (может ли он браться за него, или нет)? Процессы, на мой взгляд, кардинально бы усложнились. Так что я ввожу понятие доступа только для первичного разграничения прав в должностях.

- Employees. Опять же обращаясь к написанному выше, можно было бы сделать зависимость немного другую, но в данном случае все выполняется: транзитивной зависимости нет.

Листинг SQL

Скрипты есть на git'e; они проверялись в SQL Developer, но в SQL Plus вроде тоже все работало (гениально, с чего бы им не работать :))

Создание пользователя (файл CreateUser.sql)

Для начала подключаемся к СУБД от имени системного администратора, к контейнеру XEPDB1:

```
sqlplus sys/password@"127.0.0.1:1521/XEPDB1" as SYSDBA
```

Далее, создаем нового пользователя Infernal с квотой по меньшей мере в 100 МБ от общего табличного пространства, минимально необходимыми привилегиями (пока только на создание таблиц и подключение) и стандартными настройками tablespaces (для пользователей – users; для хранения промежуточных состояний – сортировки, подзапросов, хэширования – temp):

```
CREATE USER name
```

```
IDENTIFIED BY 1204
```

```
DEFAULT TABLESPACE users
```

```
TEMPORARY TABLESPACE temp
```

```
QUOTA 100M ON users;
```

```
GRANT CREATE SESSION TO Infernal;
```

```
GRANT CREATE TABLE TO Infernal;
```

Собственно, теперь, чтобы “попасть” в нашего пользователя, можно воспользоваться следующими вариантами:

- Через SQL Plus: зайти как раньше под sysdba в контейнер XEPDB1 и использовать

`CONNECT Infernal/1204@"127.0.0.1/XEPDB1"` (почему нужно обязательно прописывать при подключении контейнер, в дискорде до меня уже написала моя коллега из Б19-515: Listener service будет пытаться переправить нас в корневой контейнер CDB\$ROOT, где нашего пользователя нет, ибо мы его создавали в хепdb1. Да и в самом условии лабораторной на это тоже обращено внимание).

- Через SQL Developer: мне лично дебажить что-либо немного более комфортно с GUI, да и импортировать SVC прямо здесь можно, как я понял, поэтому я подключил БД в SQL Developer (и созданного пользователя, и sys, просто на всякий):

Выводы

В результате анализа и проектирования базы данных на основе выбранной легенды был предложен вариант реализации, доказано, что данный вариант находится в минимально-необходимой нормальной форме, а так же создана данная БД в СУБД Oracle SQL с помощью утилиты SQL Developer. Получены практические навыки работы с СУБД а также заложена основа для дальнейшей работы.