Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Базы данных (БД)

ОТЧЕТ

по лабораторной работе № 2

Тема работы: Доработка модели базы данных

Выполнил

студент: гр. 151003 Барановский Р.А.

Проверил: Фадеева Е.Е.

Минск 2023

**ИТОГОВАЯ МОДЕЛЬ С SQL-СКРИПТОМ НАХОДИТСЯ В КОНЦЕ ОТЧЕТА!**

Исходная инфологическая модель:

* Account – *Учётная запись (описывает учётную запись)*:
  + id *(идентификатор аккаунта)*;
  + balance (*баланс счёта*, MONEY);
  + account\_owner (*владелец счёта*, FK);
  + system\_account (*флаг, указывающий, что эта учётная запись не принадлежит человеку*).
* Status – *Статус (статус аккаунта, например, «Активен», «Заблокирован» и т.д.)*:
  + id *(идентификатор статуса)*;
  + name (*название статуса*).
* Transaction operational – *Текущие транзакции (для транзакций в текущем месяце)*:
  + id (*идентификатор транзакции*);
  + source\_account (*исходный счёт*, FK);
  + destination\_account (*целевой счёт,* FK);
  + date\_and\_time (*дата и время транзакции*);
  + sum (*общая сумма транзакции*).
* Transaction archive – *Архив транзакций (для транзакций до текущего месяца)*:
  + id (*идентификатор транзакции*);
  + source\_account (*исходный счёт*, FK);
  + destination\_account (*целевой счёт,* FK);
  + date\_and\_time (*дата и время транзакции*);
  + sum (*общая сумма транзакции*).
* Account owner – *Владелец счёта (клиент банка)*:
  + id (*идентификатор владельца аккаунта*);
  + name (*имя владельца аккаунта*).
* Site page – *Страница сайта (страница сайта банка)*:
  + id (*идентификатор страницы*);
  + parent\_page (*родительская страница,* FK);
  + name (*название страницы*).
* Office – *Офис* (*офис банка*):
  + id (*идентификатор офиса*);
  + city (*местонахождение офиса*);
  + name (*название офиса*);
  + total\_sells\_sum (*сумма прибыли офиса*, MONEY).

Исходная даталогическая модель:

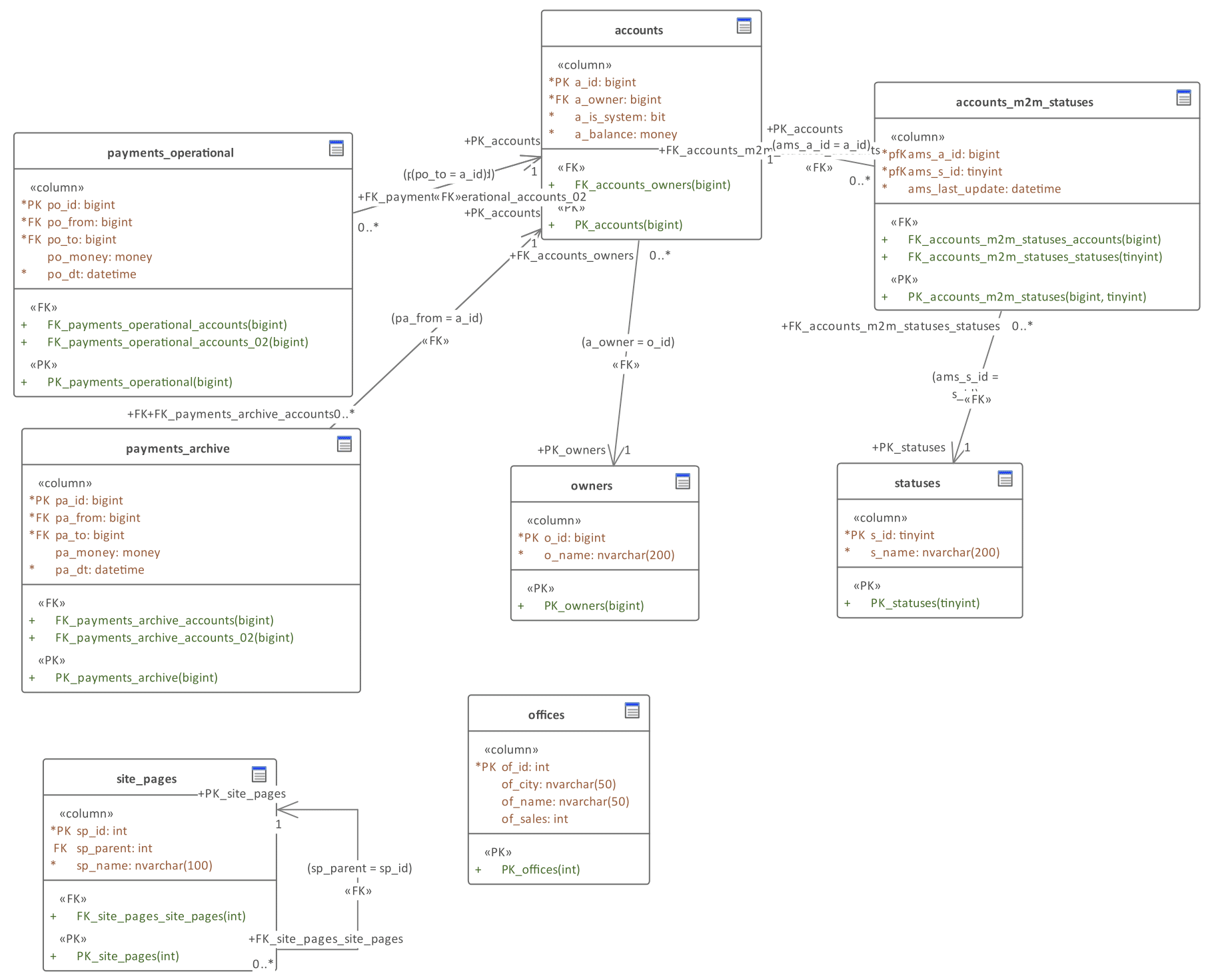


Рисунок – Даталогическая модель

Измененная даталогическая модель:

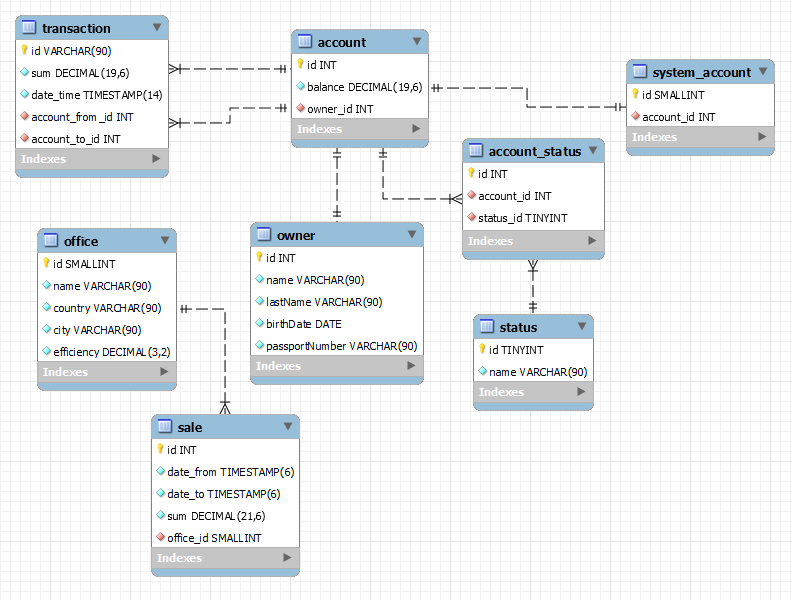


Рисунок 2 – Измененная даталогическая модель

Вариант 1: Работа банка

Ответы для исходной модели

1. Какие каскадные операции необходимы в этой базе данных? Опишите их.

**Ответ:**

Следует добавить каскадную операцию обновления для всех FK, ссылающихся на account, чтобы в случае обновления PK аккаунта производилось обновление FK в таблицах payments\_operational, payments\_archive, accounts\_m2m\_statuses и, в зависимости от требований к системе, каскадную операцию удаления или, если необходимо сохранять информацию о транзакциях аккаунта даже после его удаления(что более вероятно, так как это может понадобиться, допустим, для отчета в налоговую), то следует добавить каскадную установку значений по умолчанию(чтобы можно было, например, идентифицировать владельца транзакции в случае необходимости после удаления аккаунта). Для FK a\_owner в таблице accounts следует добавить каскадное обновление и каскадное удаление(при удалении владельца аккаунта сам аккаунт обязан быть удален). Также следует добавить каскадное обновление/удаление для FK FK\_accounts\_m2m\_statuses в accounts\_m2m\_statuses для удаления статуса у аккаунта при удалении статуса как такового и изменения FK при изменении PK.

1. Существует ли возможность аномалий операций вставки, обновления, удаления данных? Составьте список таких случаев и внесите в базу данных исправления, позволяющие избежать таких аномалий.

**Ответ:**

В таблице offices присутствуют аномалии удаления и модификации из-за поля of\_city. Аномалия удаления обусловлена тем, что поле of\_city может иметь значение определенного города только в одном объекте, при удалении объекта будет удалено также любое упоминание о данном городе. Аномалия модификации обусловлена тем, что при изменении данных в таблице offices нам может понадобится выбрать конкретные офисы по значению атрибута of\_city, при этом данный атрибут может принимать разные значения при обозначении одного и того же города, например «Минск» и «Minsk», что приведет к аномалии. В качестве решения создадим отдельную таблицу city и добавим соответствующий FK в таблицу offices. Даталогическая модель представлена на рисунке 5.

1. Можно ли использовать схемы «звезда» или «снежинка» с этой базой данных, чтобы избежать некоторых аномалий операций с данными? Переработайте схему, сравните новую с исходной и составьте список аномалий работы с данными, которые были устранены (или, наоборот, появились).

**Ответ:**

Звезда:

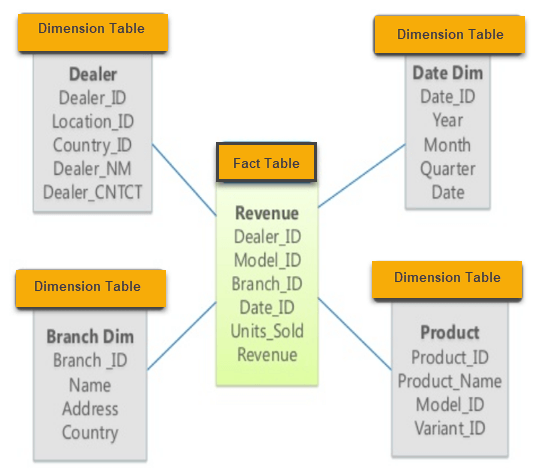


Рисунок – Схема «звезда»

Снежинка:

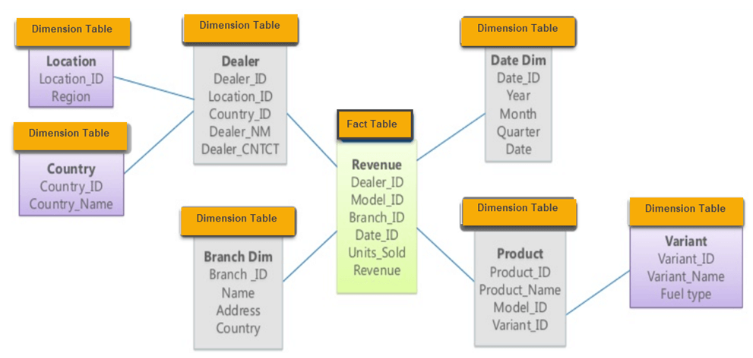


Рисунок – Схема «Снежинка»

Использование звезды потребует существенной денормализации БД, в связи с чем не стоит использовать ее в данном случае. К тому же ее использование невозможно из-за наличия one-to-many связей между accounts и payments\_operational, accounts и payments\_archive, accounts и accounts\_m2m\_statuses. Использование снежинки также невозможно из-за наличия one-to-many связей, а изменить их каким-либо образом не представляется возможным. Например, попробуем изменить таблицы accounts, statuses, accounts\_m2m\_statuses. Уберем таблицу accounts\_m2m\_statuses, затем можем или сделать FK в accounts на statuses, или создать VARCHAR атрибут в accounts. В обоих случаях аккаунт не будет иметь больше одного статуса, а во втором случае так и вовсе возникнет аномалия модификации и удаления, а возможно и вставки, если аккаунт может не иметь статуса.

1. Составьте список всех функциональных зависимостей в базе данных.

**Ответ:**

Функциональная зависимость определяется как зависимость неключевого атрибута от значения составного первичного ключа или же зависимость неключевого атрибута от первичного ключа и ключевых атрибутов(если же некоторый неключевой атрибут функционально зависит от другого неключевого атрибута, а тот, например, от PK, то произойдет нарушение 3NF). Данная зависимость может быть полной(неключевой атрибут зависит от всех частей составного ключа или от первичного ключа и всех ключевых атрибутов) и неполной(неключевой атрибут зависит не от всех частей составного ключа или только первичного ключа, но не всех ключевых атрибутов). В исходной модели отсутствуют ключевые атрибуты в каких-либо таблицах, присутствуют лишь PK. Все неключевые атрибуты при этом функционально зависят от PK, это требуется для соблюдения 2-ой нормальной формы. Исключение составляет атрибут of\_sales в таблице offices. Он не зависит функционально от of\_id.

1. Существуют ли отношения, имеющие многозначные зависимости? Если «да», как можно переработать схему, чтобы избежать таких зависимостей?

**Ответ:**

Ни в исходной модели, ни в измененной нет многозначных зависимостей. По определению: многозначные зависимости – зависимости, которые образуются, если имеется 3+ атрибута, причем добавление новой записи будет вести к добавление нескольких записей, которые представляют собой кросс-комбинацию.

1. Нарушает ли схема какие-либо «требования нормализации»? Если «да», доработайте схему, чтобы избежать таких нарушений.

**Ответ:**

1NF – соблюдена, так как каждый кортеж содержит ровно одно значение в каждом своем атрибуте и существующие поля не поддаются разделению.

2NF – не соблюдена, в таблице offices поле of\_sales не зависит от PK of\_id. Прибыль зависит от некоторых внутренних показателей офиса, но не от того, какой это конкретно офис. Можно привести аналогию с группами студентов. Количество лет обучения зависит от года поступления(некоторая характеристика группы), но никак не от номера группы, так же и прибыль офиса зависит от некоторой внутренней характеристики(показателей), но не от его id(номера). Данная проблема разрешена в измененной модели.

1. Существуют ли какие-либо потенциальные проблемы с производительностью базы данных? Если «да», запишите их.

**Ответ:**

Существуют. Вероятное отсутствие индексов снижает скорость поиска в БД, хоть и ускоряет добавление новых данных(стоит отметить, что в некоторых таблицах это является преимуществом, а в некоторых недостатком. Например, добавление новых акканутов вероятнее всего будет производиться существенно реже, чем поиск аккаунтов, что делает разумным добавление индексов в таблицу accounts. В то же время добавление транзакций вероятно будет происходить чаще, чем их чтение, что делает добавление индексов нерациональным). К тому же для данной модели необходимо добавлять каскады, так как в банковской системе критически важно, чтобы во всех таблицах хранились актуальные данные, а каскады снижают производительность.

1. Для каждого отношения в базе данных определите, в какой нормальной форме оно находится. Запишите ответ.

**Ответ:**

accounts – 5NF

payments\_operational – 5NF

payments\_archive – 5NF

site\_pages – 5NF

owners – 5NF

offices – 1NF

accounts\_m2m\_statuses – 5NF

1. Есть ли отношения с возможной, но ненужной дальнейшей нормализацией? Составьте список.

**Ответ:**

В таблице offices атрибут of\_sales не удовлетворяет требованияем 2NF, поэтому следует создать отдельную таблицу, что и было сделано в измененной модели.

1. Можно ли добиться некоторого повышения производительности за счёт денормализации схемы? Обоснуйте своё мнение.

**Ответ:**

Можно, денормализация приведет к уменьшению количества таблиц в схеме. За счет этого уменьшится количество JOIN-запросов, количество каскадов, которые необходимо добавить. Это приведет к повышению производительности.

1. Можно ли добиться некоторого повышения производительности, добавив в схему кэширующие отношения? Обоснуйте своё мнение.

**Ответ:**

Можно, например, для таблицы accounts добавить атрибут transactions, где будут содержаться все производенные с аккаунтом транзакции, что ускорит их поиск. Дополнительным бонусом станет то, что не придется добавлять индексы в transactions, куда, вероятнее всего, будут очень часто добавляться записи.

1. Добавьте в БД все необходимые индексы, представления, хранимые процедуры и т.д.

**Ответ:**

Помимо добавленных автоматически индексов для всех PK, FK, а также UNIQUE атрибутов можно добавить следующие индексы:

- idx\_po\_dt для атрибута po\_dt в таблице payments\_operational. Это позволит быстрее производить поиск текущих транзакций по дате и времени.

- idx\_po\_dt для атрибута po\_dt в таблице payments\_archive. Это позволит быстрее производить поиск архивных транзакций по дате и времени.

- idx\_of\_city\_of\_sales для атрибутов (of\_city, of\_sales) в таблице office. Это позволит быстрее производить поиск офисов по городу, а также по городу и прибыли(например, для составления отчетности по региону).

Ответы для измененной модели

1. Какие каскадные операции необходимы в этой базе данных? Опишите их.

**Ответ:**

Для измененной модели, учитывая, что структура в таблицах, где возможны каскадные операции, аналогична структуре таблиц исходной модели, каскадные операции остаются теми же. При этом в измененной модели присутствуют 2 дополнительные таблицы system\_account и sale. Соответственно необходимо добавить каскадное обновление/удаление для FK office\_id в sale и каскадное обновление/удаление для FK account\_id в system\_account. В итоговой модели на рисунке 5 также добавились FK office\_id, location\_id, country\_id, city\_id. К данным FK также добавляем каскадное обновление/удаление.

1. Существует ли возможность аномалий операций вставки, обновления, удаления данных? Составьте список таких случаев и внесите в базу данных исправления, позволяющие избежать таких аномалий.

**Ответ:**

Аномалии удаления и модификации присутствуют в таблице office в атрибутах country и city, которой соответствует таблица offices в исходной модели без дополнительных атрибутов. Решение аналогично предложенному в пункте 2 для исходной модели. Создадим таблицу country и city с связью many-to-many(в разных странах могут находиться города с одинаковым названием), а FK в таблице office будет указывать на связующую таблицу, определяя местоположение офиса. Даталогическая модель представлена на рисунке 5.

1. Можно ли использовать схемы «звезда» или «снежинка» с этой базой данных, чтобы избежать некоторых аномалий операций с данными? Переработайте схему, сравните новую с исходной и составьте список аномалий работы с данными, которые были устранены (или, наоборот, появились).

**Ответ:**

Проблемы использования данных схем в измененной модели аналогичны проблемам их использования в исходной. Вообще итоговая схема наиболее напоминает схему «галактика», где имеется несколько таблиц фактов с соответствующими таблицами измерений.

1. Составьте список всех функциональных зависимостей в базе данных.

**Ответ:**

Справедливо то же, что и для исходной модели. Однако теперь в таблице office отсутствует атрибут sale, что позволяет утверждать, что теперь все неключевые атрибуты полностью функционально зависят от PK.

1. Существуют ли отношения, имеющие многозначные зависимости? Если «да», как можно переработать схему, чтобы избежать таких зависимостей?

**Ответ:**

Ни в исходной модели, ни в измененной нет многозначных зависимостей. По определению: многозначные зависимости – зависимости, которые образуются, если имеется 3+ атрибута, причем добавление новой записи будет вести к добавление нескольких записей, которые представляют собой кросс-комбинацию.

1. Нарушает ли схема какие-либо «требования нормализации»? Если «да», доработайте схему, чтобы избежать таких нарушений.

**Ответ:**

1NF – соблюдена, так как каждый кортеж содержит ровно одно значение в каждом своем атрибуте и существующие поля не поддаются разделению.

2NF – соблюдена в сильной форме, так как в каждой таблице каждый неключевой атрибут полностью функционально зависит от PK и в то же время каждый неключевой атрибут полностью функционально зависит от любого ключевого атрибута сущности.

3NF/BCNF – соблюдена, отсутствуют транзитивные зависимости каких-либо неключевых(3NF)/ключевых и неключевых(BCNF) атрибутов от первичного ключа. Несоблюдение данной нормальной формы зачастую можно выявить находя многочисленное дублирование информации в таблице(к слову, несмотря на полезность данного приема, всегда стоит анализировать таблицу в соответствии с оригинальной формулировкой соответствия таблицы 3NF, иначе существует риск прийти к неправильному выводу. Например, в таблице office есть поля city и country, и при нахождении в одном городе и стране множества офисов информация будет дублироваться, однако здесь нет нарушения 3NF, т.к. транзитивная зависимость отсутствует, а присутствует лишь функциональная между id и city, country).

4NF – по определению: переменная отношения находится в 4-ой нормальной форме, если она удовлетворяет BCNF и любая многозначная зависимость, содержащаяся в этом отношении, определяется каким-то из супер-ключей. Однако в измененной модели отсутствуют многозначные зависимости, следовательно соблюдена 4NF.

5NF – упрощенно: необходимо знать все потенциальные ключи и зависимости соединения в отношении и необходимо убедиться, что каждая проекция для каждой зависимости соединения содержит в себе потенциальный ключ. Соблюдена.

DKNF – упрощенно: любые правила, применимые к отношению, проистекают из свойств атрибутов, а не из некоторых “скрытых правил”. Иначе говоря, любые ограничения, которые мы хотим применить в отношении, должны задаваться лишь доменными и ключевыми ограничениями, а не “скрытыми” правилами. Здесь существуют некоторые сложности. Отнесение к данной нормальной форме требует очень подробного знания предметной области. Например, в силу постановки задачи до конца неизвестно, о каких статусах идет речь в таблице status. Может, это вспомогательная информация об аккаунте (например, аккаунт является корпоративным или личным, владеет аккаунтом юр.лицо или физ.лицо и др.), а может это информация о статусе аккаунта в системе(действующий, забанен, заморожен и др.). Во втором случае возможно наличие лишь одного статуса у аккаунта, в первом множество. Данный момент как раз и является теми самыми “скрытыми правилами”, наличие которых при такой структуре не позволяет задавать ограничения отношений только доменными и ключевыми ограничениями. В примере при такой структуре таблиц максимум, который мы можем получить, если хотим иметь и вспомогательную информацию об аккаунте, и о его статусе в системе с учетом поставленных ограничений, может быть достигнут разве что за счет использования некоторого “магического” триггера, который не позволит добавить к одному аккаунту несколько состояний в системе. А в качестве решения следовало бы разбить таблицу status на таблицы status и info. В связи с вышесказанным и приведенным примером, DKNF в измененной модели нельзя считать соблюденной.

6NF – не соблюдена в связи с несоблюдением DKNF. Вообще 6-ая форма является обобщением 5-ой для хронологических баз данных.

1. Существуют ли какие-либо потенциальные проблемы с производительностью базы данных? Если «да», запишите их.

**Ответ:**

Для измененной модели актуально то же, что и для исходной. К тому же добавляется дополнительная проблема: в связи с увеличением количества таблиц(декомпозицией) произойдет увеличение количества JOIN-запросов, которые плохо влияют на производительность.

1. Для каждого отношения в базе данных определите, в какой нормальной форме оно находится. Запишите ответ.

**Ответ:**

Account – 5NF

Transaction – 5NF

System\_account – 5NF

Account\_status – 5NF

Owner – 5NF

Office – 5NF

Sale – 5NF

Status – 5NF

1. Есть ли отношения с возможной, но ненужной дальнейшей нормализацией? Составьте список.

**Ответ:**

Стоит отдельно поговорить про 6NF, так как про нее мало сказано в пункте 6. Некоторые отношения можно вертикально декомпозировать, однако лишь в таблицах sale и transaction присутствуют зависящие от временного параметра атрибуты, поэтому приведение к 6NF не обязательно, так как отделение этих атрибутов не будет стоить времени выполнения дополнительных JOIN-запросов.

1. Можно ли добиться некоторого повышения производительности за счёт денормализации схемы? Обоснуйте своё мнение.

**Ответ:**

Ответ аналогичен пункту 10 для исходной модели.

1. Можно ли добиться некоторого повышения производительности, добавив в схему кэширующие отношения? Обоснуйте своё мнение.

**Ответ:**

Для таблицы account применимо описанное в 11 пункте для исходной модели. Также для ускорения получения полной финансовой отчетности офиса можно добавить атрибут sales, где будут содержаться финансовые отчеты офиса, в таблицу office, т.к. различных финансов отчетов у офиса может быть крайне много.

1. Добавьте в БД все необходимые индексы, представления, хранимые процедуры и т.д.

**Ответ:**

Помимо добавленных автоматически индексов для всех PK, FK, а также UNIQUE атрибутов можно добавить следующие индексы:

- idx\_date\_time для атрибута date\_time в таблице transaction. Это позволит быстрее производить поиск транзакций по дате и времени.

- idx\_country\_сity\_efficiency для атрибутов (country, city, efficiency) в таблице office. Это позволит быстрее производить поиск офиса по стране, стране и городу, позволит быстрее находить эффективность офисов, находящихся в определенной стране и определенном городе (в итоговом варианте это будет составной индекс idx\_location\_id\_efficiency (location\_id, efficiency)).

- idx\_balance для атрибута balance в таблице account. Это позволит быстрее сортировать аккаунты по балансу и соответственно вводить условия пользования аккаунтом, делать предложения, акции в зависимости от баланса.

Представления не добавляются, однако при необходимости можно добавить, например, представление для таблицы owner и ограничить доступ к таблице owner, что позволит получать доступ к персональным данным владельцев аккаунтов лишь через виртуальную таблицу, которую можно будет неким образом настроить и тем самым повысить безопасность.

В хранимые процедуры предлагается добавить 2:

- «full\_account\_owner\_info» для возврата полной информации об аккаунте и его владельце по id аккаунта

Пример работы:





- «full\_office\_address\_info» для возврата полной информации о местонахождении офиса по id офиса

Пример работы:





**Итог:**

**Итоговая даталогическая модель:**

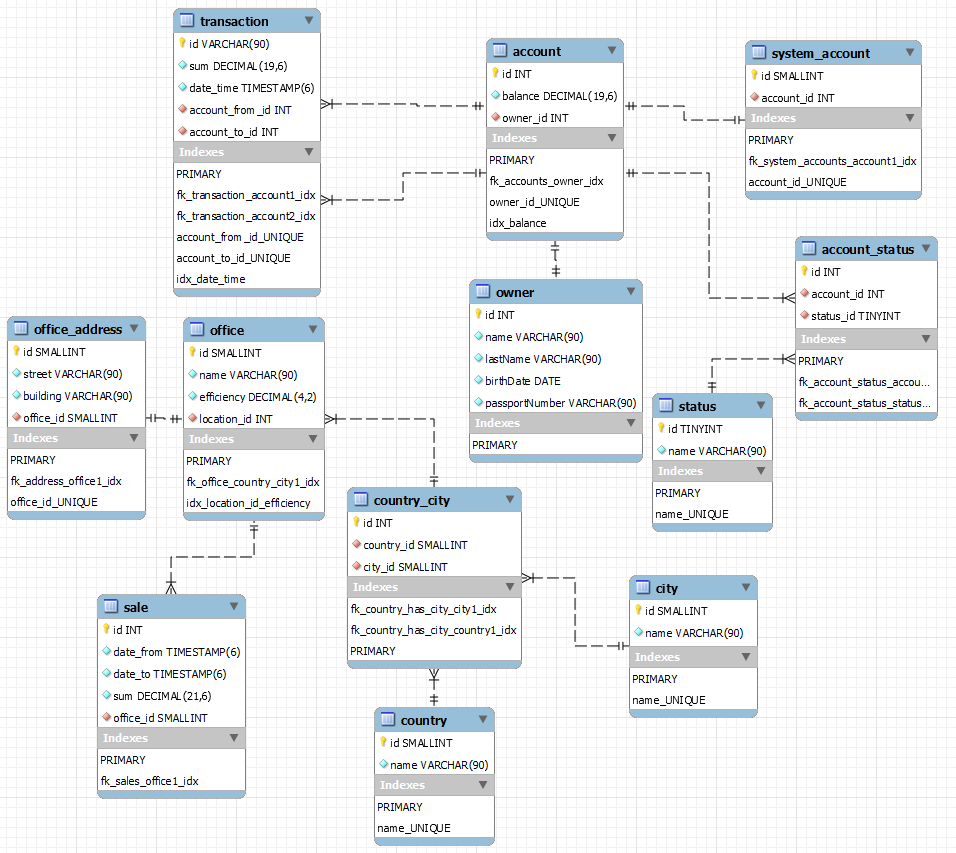


Рисунок – Итоговая даталогическая модель

**SQL-скрипт:**

-- MySQL Script generated by MySQL Workbench

-- Sun Oct 15 13:04:10 2023

-- Model: New Model Version: 1.0

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

DROP SCHEMA IF EXISTS `mydb` ;

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;

USE `mydb` ;

-- -----------------------------------------------------

-- Table `mydb`.`owner`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`owner` ;

CREATE TABLE IF NOT EXISTS `mydb`.`owner` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(90) NOT NULL,

`lastName` VARCHAR(90) NOT NULL,

`birthDate` DATE NOT NULL,

`passportNumber` VARCHAR(90) NOT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`account`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`account` ;

CREATE TABLE IF NOT EXISTS `mydb`.`account` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`balance` DECIMAL(19,6) NOT NULL,

`owner\_id` INT UNSIGNED NOT NULL,

PRIMARY KEY (`id`),

INDEX `fk\_accounts\_owner\_idx` (`owner\_id` ASC) VISIBLE,

UNIQUE INDEX `owner\_id\_UNIQUE` (`owner\_id` ASC) VISIBLE,

INDEX `idx\_balance` (`balance` ASC) VISIBLE,

CONSTRAINT `fk\_accounts\_owner`

FOREIGN KEY (`owner\_id`)

REFERENCES `mydb`.`owner` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`system\_account`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`system\_account` ;

CREATE TABLE IF NOT EXISTS `mydb`.`system\_account` (

`id` SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

`account\_id` INT UNSIGNED NOT NULL,

PRIMARY KEY (`id`),

INDEX `fk\_system\_accounts\_account1\_idx` (`account\_id` ASC) VISIBLE,

UNIQUE INDEX `account\_id\_UNIQUE` (`account\_id` ASC) VISIBLE,

CONSTRAINT `fk\_system\_accounts\_account1`

FOREIGN KEY (`account\_id`)

REFERENCES `mydb`.`account` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`status`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`status` ;

CREATE TABLE IF NOT EXISTS `mydb`.`status` (

`id` TINYINT UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(90) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `name\_UNIQUE` (`name` ASC) VISIBLE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`account\_status`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`account\_status` ;

CREATE TABLE IF NOT EXISTS `mydb`.`account\_status` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`account\_id` INT UNSIGNED NOT NULL,

`status\_id` TINYINT UNSIGNED NOT NULL,

PRIMARY KEY (`id`),

INDEX `fk\_account\_status\_account1\_idx` (`account\_id` ASC) VISIBLE,

INDEX `fk\_account\_status\_status1\_idx` (`status\_id` ASC) VISIBLE,

CONSTRAINT `fk\_account\_status\_account1`

FOREIGN KEY (`account\_id`)

REFERENCES `mydb`.`account` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk\_account\_status\_status1`

FOREIGN KEY (`status\_id`)

REFERENCES `mydb`.`status` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`transaction`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`transaction` ;

CREATE TABLE IF NOT EXISTS `mydb`.`transaction` (

`id` VARCHAR(90) NOT NULL,

`sum` DECIMAL(19,6) UNSIGNED NOT NULL,

`date\_time` TIMESTAMP(6) NOT NULL,

`account\_from\_id` INT UNSIGNED NOT NULL,

`account\_to\_id` INT UNSIGNED NOT NULL,

PRIMARY KEY (`id`),

INDEX `fk\_transaction\_account1\_idx` (`account\_from\_id` ASC) VISIBLE,

INDEX `fk\_transaction\_account2\_idx` (`account\_to\_id` ASC) VISIBLE,

UNIQUE INDEX `account\_from\_id\_UNIQUE` (`account\_from\_id` ASC) VISIBLE,

UNIQUE INDEX `account\_to\_id\_UNIQUE` (`account\_to\_id` ASC) VISIBLE,

INDEX `idx\_date\_time` (`date\_time` ASC) VISIBLE,

CONSTRAINT `fk\_transaction\_account1`

FOREIGN KEY (`account\_from\_id`)

REFERENCES `mydb`.`account` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk\_transaction\_account2`

FOREIGN KEY (`account\_to\_id`)

REFERENCES `mydb`.`account` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`country`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`country` ;

CREATE TABLE IF NOT EXISTS `mydb`.`country` (

`id` SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(90) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `name\_UNIQUE` (`name` ASC) VISIBLE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`city`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`city` ;

CREATE TABLE IF NOT EXISTS `mydb`.`city` (

`id` SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(90) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `name\_UNIQUE` (`name` ASC) VISIBLE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`country\_city`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`country\_city` ;

CREATE TABLE IF NOT EXISTS `mydb`.`country\_city` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`country\_id` SMALLINT UNSIGNED NOT NULL,

`city\_id` SMALLINT UNSIGNED NOT NULL,

INDEX `fk\_country\_has\_city\_city1\_idx` (`city\_id` ASC) VISIBLE,

INDEX `fk\_country\_has\_city\_country1\_idx` (`country\_id` ASC) VISIBLE,

PRIMARY KEY (`id`),

CONSTRAINT `fk\_country\_has\_city\_country1`

FOREIGN KEY (`country\_id`)

REFERENCES `mydb`.`country` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk\_country\_has\_city\_city1`

FOREIGN KEY (`city\_id`)

REFERENCES `mydb`.`city` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`office`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`office` ;

CREATE TABLE IF NOT EXISTS `mydb`.`office` (

`id` SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(90) NOT NULL,

`efficiency` DECIMAL(4,2) UNSIGNED NOT NULL,

`location\_id` INT UNSIGNED NOT NULL,

PRIMARY KEY (`id`),

INDEX `fk\_office\_country\_city1\_idx` (`location\_id` ASC) VISIBLE,

INDEX `idx\_location\_id\_efficiency` (`location\_id` ASC, `efficiency` ASC) VISIBLE,

CONSTRAINT `fk\_office\_country\_city1`

FOREIGN KEY (`location\_id`)

REFERENCES `mydb`.`country\_city` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`sale`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`sale` ;

CREATE TABLE IF NOT EXISTS `mydb`.`sale` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`date\_from` TIMESTAMP(6) NOT NULL,

`date\_to` TIMESTAMP(6) NOT NULL,

`sum` DECIMAL(21,6) UNSIGNED NOT NULL,

`office\_id` SMALLINT UNSIGNED NOT NULL,

PRIMARY KEY (`id`),

INDEX `fk\_sales\_office1\_idx` (`office\_id` ASC) VISIBLE,

CONSTRAINT `fk\_sales\_office1`

FOREIGN KEY (`office\_id`)

REFERENCES `mydb`.`office` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `mydb`.`office\_address`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `mydb`.`office\_address` ;

CREATE TABLE IF NOT EXISTS `mydb`.`office\_address` (

`id` SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

`street` VARCHAR(90) NOT NULL,

`building` VARCHAR(90) NOT NULL,

`office\_id` SMALLINT UNSIGNED NOT NULL,

PRIMARY KEY (`id`),

INDEX `fk\_address\_office1\_idx` (`office\_id` ASC) VISIBLE,

UNIQUE INDEX `office\_id\_UNIQUE` (`office\_id` ASC) VISIBLE,

CONSTRAINT `fk\_address\_office1`

FOREIGN KEY (`office\_id`)

REFERENCES `mydb`.`office` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

**Результат работы скрипта:**

