Раздел 4: Проектирование баз данных

4.1. Проектирование на инфологическом уровне

4.1.1. Цели и задачи проектирования на инфологическом уровне

Перед прочтением материала данной главы стоит повторить основы моделирования баз данных{7}.

Как и было сказано в определении инфологического уровня моделирования{9}, основной его целью является создание модели, отражающей сущности предметной области, их атрибуты и связи (возможно, пока не все) между сущностями.

Т.е. необходимо максимально глубоко исследовать предметную область и выразить её понятия в виде сущностей, атрибутов, связей.

Исследование предметной области как раз и представляет основную сложность, разделяющуюся на следующие локальные проблемы:

- извлечение информации;
- глубина исследования;
- границы исследования.

Извлечение информации — универсальная проблема для любого проекта (даже вне контекста баз данных). Как правило[190], у заказчика нет готового полного технического описания проекта, потому необходимо организовать сбор требований к проекту, и в них выделить те требования, которые прямо или косвенно относятся к базе данных.

Существует множество техник выявления требований (интервью, работа с фокусными группами, анкетирование, семинары и мозговой штурм, наблюдение, прототипирование, анализ документов, моделирование процессов и взаимодействий и т.д. и т.п.[191])

Вне зависимости от выбранного подхода процесс будет долгим, итерационным и направленным на решение главной задачи: собрать максимально корректную информацию в полном объёме. И это приводит нас к следующей локальной проблеме.

Глубина исследования — самая большая сложность для начинающих проектировщиков, что особенно хорошо заметно на примере учебных работ. Недостаточно выявить лишь *некоторые* сущности и *некоторые* их атрибуты. Необходимо выявить их *все*.

Говоря образно, база данных *реальной* библиотеки не может состоять из таблиц «книги» (название, год выпуска) и «авторы» (ФИО, год рождения) — в такой базе данных будут десятки таблиц с десятками полей.

Проблема усложняется в силу того, что в общем случае — это не задача заказчика — продумывать все возможные детали, нюансы, аспекты. Это задача именно проектировщика базы данных. И задача весьма серьёзная, т.к. неудача в её решении почти гарантированно приведёт к тому, что база данных будет неадекватна предметной области{11}.

С другой стороны, есть опасность выйти за границы проекта и столкнуться со следующей проблемой.

---

[190] На самом деле — всегда ☺.

[191] См. раздел «2.2.3. Источники и пути выявления требований» в книге «Тестирование программного обеспечения. Базовый курс». (С.С. Куликов) [https://svyatoslav.biz/software_testing_book/]

Границы исследования — проблема часто не техническая, а управленческая, но от этого она не становится проще: в силу настояния заказчика или из технических соображений в базу данных могут начать попадать сущности, вполне реально существующие и связанные с предметной областью, но не актуальные для реализуемого проекта.

Так, продолжая пример с библиотекой, можно от самой базы данных библиотеки перейти к логистике (книги ведь как-то в библиотеку попадают), складскому учёту (книги ведь где-то хранятся), бухгалтерии (ведь финансовые вопросы надо как-то решать), кадровому учёту (ведь в библиотеке работают сотрудники) и т.д.

Чтобы избежать этого эффекта (он называется «размытие границ») применяются специальные техники, не относящиеся к тематике данной книги, однако всегда стоит удерживать в голове, базу данных *чего* мы проектируем. И поднимать вопрос о том, не вышли ли мы за границы проекта, если возникают сомнения, что те или иные сущности реального мира имеют отношение к проектируемой базе данных.

К перечню задач данного уровня проектирования также можно смело добавить обеспечение соответствия базы данных ключевым требованиям{11}: и пусть их невозможно обеспечить одним лишь проектированием на инфологическом уровне, именно здесь закладывается фундамент для многих решений, которые будут приняты на следующих уровнях и позволят создать действительно эффективную базу данных.

Задание 4.1.а: сформулируйте список вопросов, ответы на которые помогли бы вам улучшить инфологическую схему базы данных «Банк»{409}.

4.1.2. Инструменты и техники проектирования на инфологическом уровне

После прочтения предыдущей главы может сложиться впечатление, что для решения столь нетривиальных задач необходимы некие сверхсложные инструменты. Вовсе нет.

Да, непростыми являются сами техники — в первую очередь техники выявления требований{285}, но даже их сложность состоит по большей части в количестве прилагаемых усилий.

Что до инструментов, то они достаточно просты и так или иначе сводятся к записи собранной информации в удобном для восприятия виде. Причём очень важно понимать, что удобно должно быть не только проектировщику базы данных, но и представителю заказчика, с которым результаты проектирования будут многократно обсуждаться.

В плане такого удобства непревзойдёнными остаются всем понятные и привычные многоуровневые списки, которые можно составлять в любом текстовом редакторе. Первым уровнем будут идти сущности, вторым — их атрибуты (см. пример на рисунке 4.1.a).

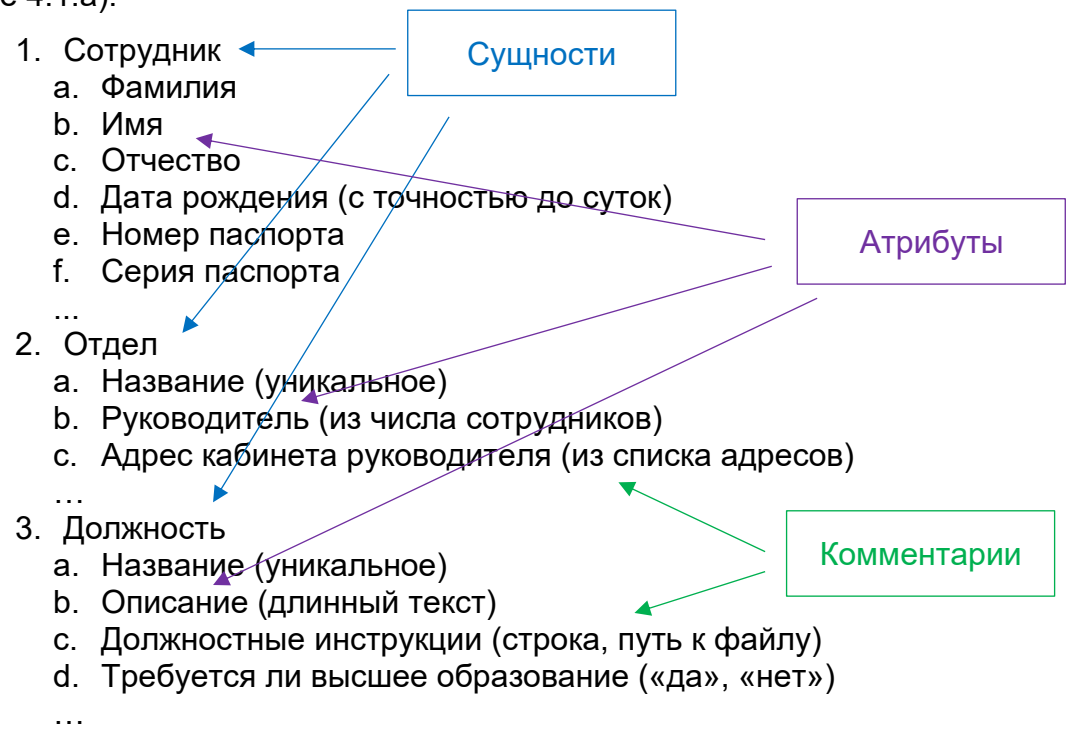


Рисунок 4.1.a — Пример представления инфологической модели в текстовом виде

Текстовое представление удобно не только своей привычностью, но и своей практичностью:

- у всех на компьютере есть необходимое программное обеспечение для просмотра и редактирования текстовых документов;
- создание и правка таких документов происходит очень быстро;
- можно очень удобно располагать пометки и комментарии (которые будут видны сразу без дополнительных действий);
- результат может быть мгновенно распечатан;
- и т.д.

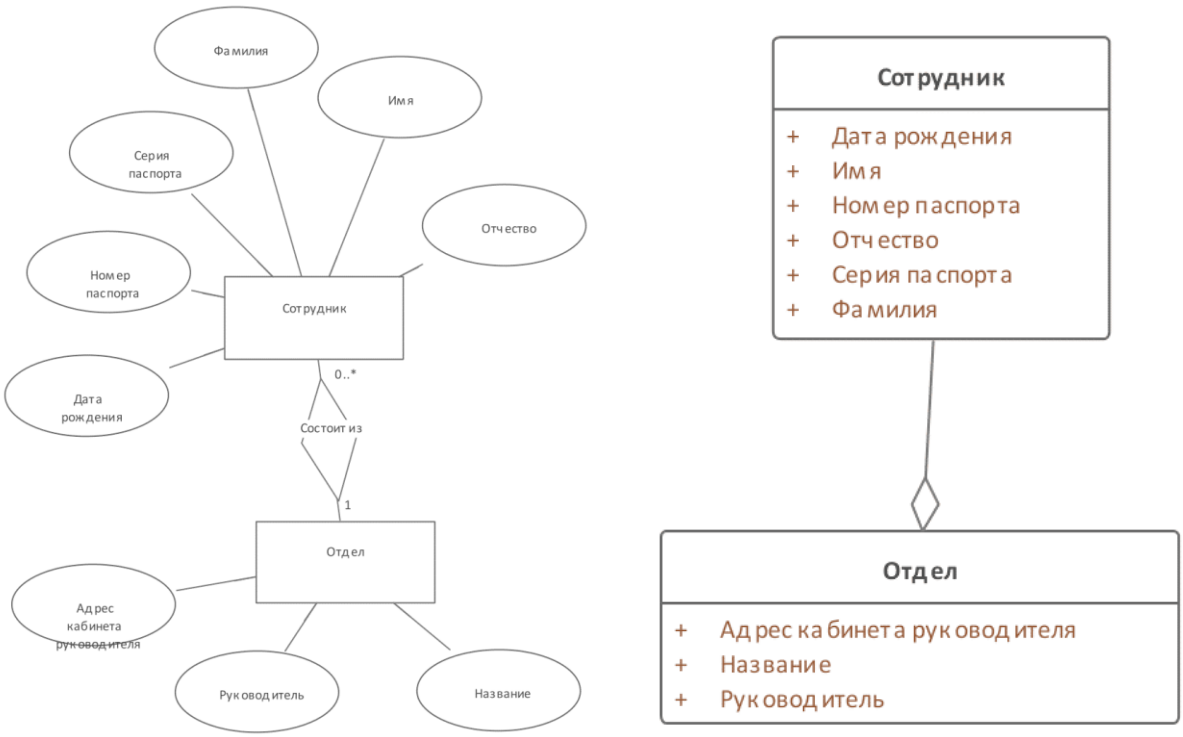
И раз всё так прекрасно, то кажется, что других инструментов не существует просто в силу ненужности. Но это не так.

У текстового представления есть ряд серьёзных недостатков:

- оно неудобно для представления взаимосвязи сущностей (фактически, это можно делать только комментариями);
- оно не компактно (может занимать несколько десятков страниц там, где другие формы представления заняли бы пару экранов);
- оно допускает разночтение технических аспектов реализации базы данных (а чаще всего эти аспекты и вовсе оказываются упущены);
- попытки устранить обозначенные выше недостатки делают текстовое представление перегруженным информацией и постепенно сводят на нет его преимущества.

Альтернативами текстовому представлению являются графические формы — в виде семантических моделей, графовых моделей и UML-диаграмм[192].

Прежде, чем мы рассмотрим их примеры, кратко скажем про упоминаемые в огромном количестве источников ER-диаграммы (entity-relation, сущность-связь). Увы, несмотря на свою теоретическую красоту, они крайне неудобны. Сравните (см. рисунок 4.1.b) представление небольшого фрагмента схемы базы данных в виде ER-диаграммы[193] (в нотации Чена) и в виде UML-диаграммы.



ER-диаграмма UML-диаграмма

Рисунок 4.1.b — Сравнение ER-диаграммы и UML-диаграммы

[192] Да, существуют специальные языки описания инфологических моделей. Например, ЯИМ (язык инфологического моделирования), но они гармонично сочетают в себе недостатки классического «списочного» подхода и сложность графических моделей. У таких языков есть свои области применения, они имеют право на существование, но назвать их широко распространёнными нельзя, потому в данной книге мы их также не рассматриваем.

[193] Даже такое мощнейшее средство проектирования как Sparx Enterprise Architect до сих пор «не умеет» корректно выравнивать надписи внутри элементов ER-диаграмм, что тоже говорит о «востребованности» такой формы представления модели.

Теперь вернёмся к тому, что действительно применяется на практике.

Семантические и графовые модели распространены не так сильно, как UML-диаграммы, но оказываются очень полезными при описании сложных взаимосвязей в предметных областях.

Их сложно использовать «напрямую» для проектирования баз данных (они не проецируются напрямую на реляционную модель), но в качестве постоянно доступной «шпаргалки» они почти незаменимы.

Допустим, нам нужно отразить в базе данных сложную взаимосвязь между ролями сотрудников (для написания контролирующих триггеров{351}). Чтобы не держать такую взаимосвязь в уме, можно выразить её семантической схемой — см. рисунок 4.1.c.

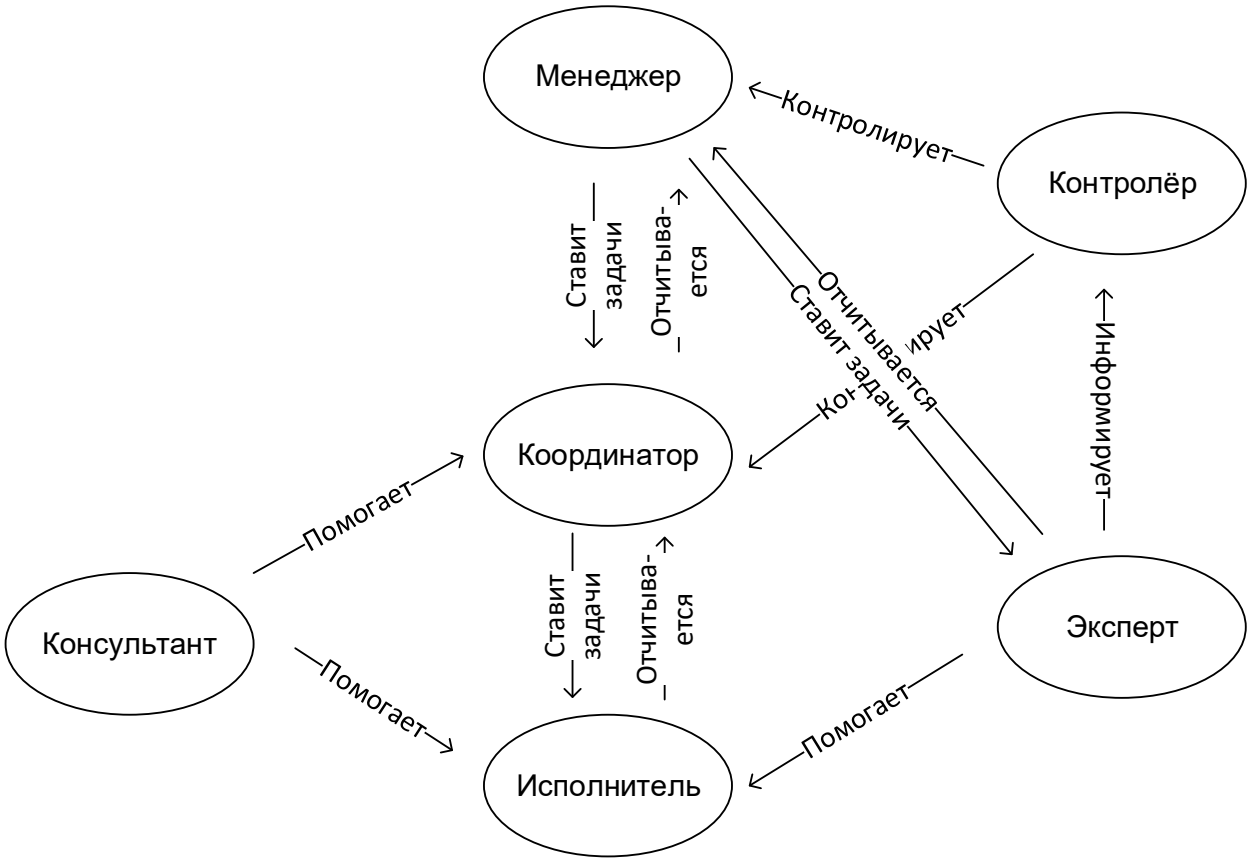


Рисунок 4.1.c — Пример семантической схемы

Графовые модели очень удобны для описания состояний и переходов между ними. Опять же, намного проще нарисовать схему (см. рисунок 4.1.d), чем постоянно держать это в уме.

Семантические и графовые модели могут быть полезны не только для описания сложных и необычных предметных областей, но и таких, в которых всё кажется привычным и очевидным, однако в данном конкретном проекте имеет некие нетипичные особенности (например, на рисунке 4.1.c показано, что менеджеру консультант не помогает, хотя интуитивно кажется, что должен помогать; на рисунке 4.1.d некая задача не может переходить между состояниями «В работе» и «Отклонено», хотя интуитивно и тут не видится никаких объективных запретов).

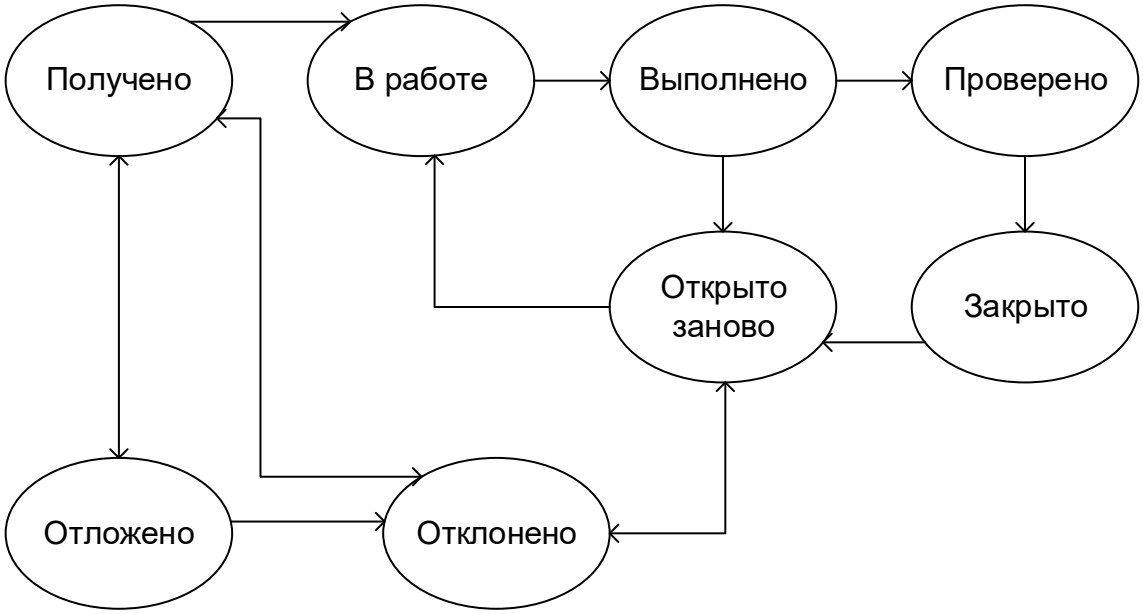


Рисунок 4.1.d — Пример графовой схемы

Наконец переходим к UML и начнём с краткого напоминания видов связей[194] (их общий перечень показан на рисунке 4.1.e). Да, возможности UML значительно шире, но для моделирования баз данных в общем случае достаточно будет помнить, что такое «связь», «класс», «атрибут», «метод».

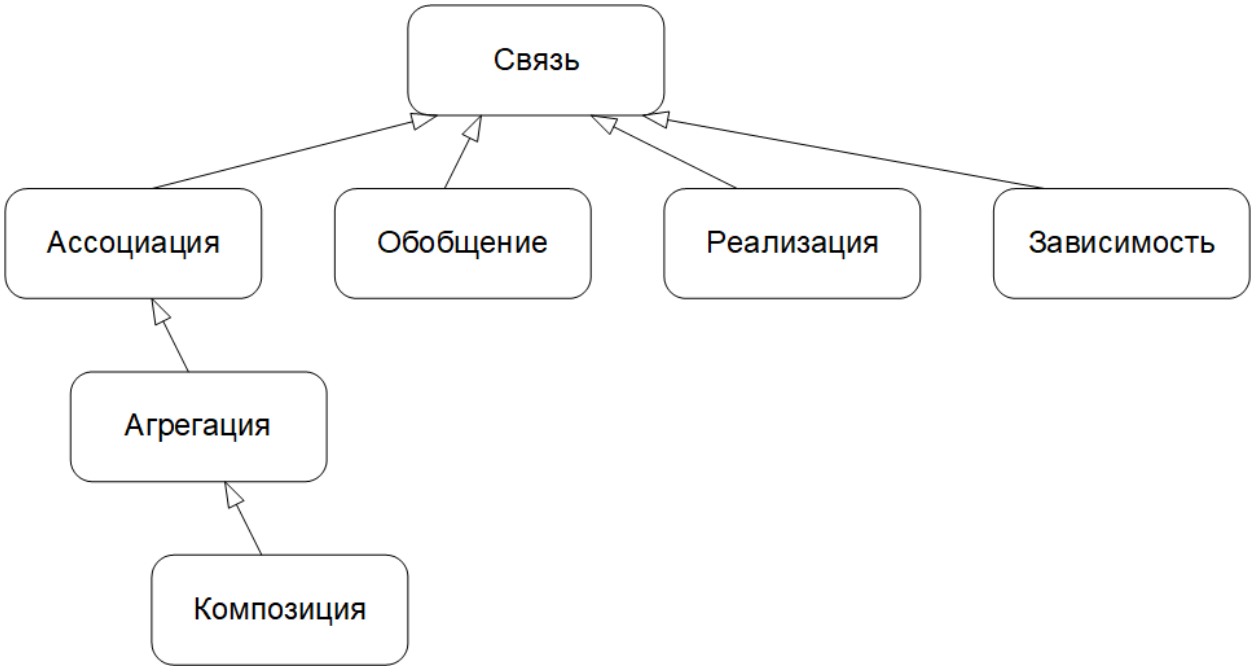


Рисунок 4.1.e — Иерархия UML-связей

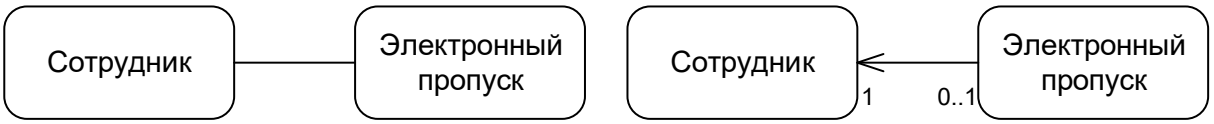
[194] Полноценное рассмотрение данной технологии выходит за рамки этой книги, но в Интернет есть большое количество хорошей документации. Начать можно отсюда: https://www.tutorialspoint.com/uml/

Рассмотрим подробнее все виды UML-связей и их применение в моделировании баз данных.

Ассоциация — самый общий, универсальный и «ни к чему не обязывающий вариант»: просто отражается тот факт, что некие сущности находятся во взаимосвязи (при этом вид и особенности этой взаимосвязи не конкретизируются, хоть при желании и можно указать некоторые параметры — например, мощность связи и её направление).

В примере ниже (рисунок 4.1.f) показано, что сотрудник и электронный пропуск связаны между собой. В примере с уточнёнными параметрами показано, что:

- электронный пропуск принадлежит сотруднику (а не наоборот);
- электронный пропуск обязан принадлежать ровно одному сотруднику;
- у сотрудника может не быть электронного пропуска;
- у сотрудника может быть не более одного электронного пропуска.



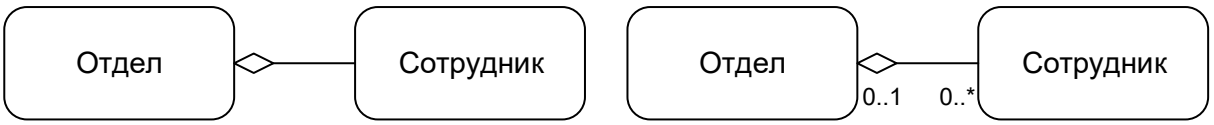
Самый общий вид — Уточнение некоторых параметров

Рисунок 4.1.f — Пример ассоциации

Агрегация — частный случай ассоциации, показывающий, что дочерние элементы входят в состав родительского элемента (но могут и существовать отдельно).

В примере ниже (рисунок 4.1.g) показано, что отдел состоит из сотрудников («агрегирует» собой сотрудников). В примере с уточнёнными параметрами показано, что:

- в отделе может быть от нуля до бесконечности сотрудников;
- сотрудник может не принадлежать никакому отделу;
- сотрудник может принадлежать не более, чем одному отделу.



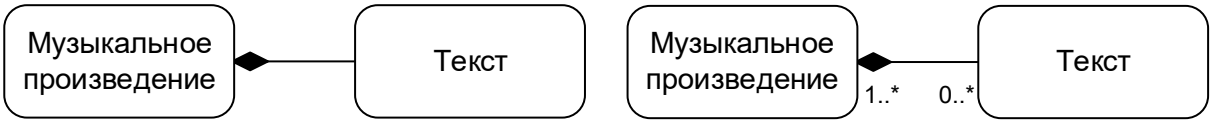
Самый общий вид — Уточнение некоторых параметров

Рисунок 4.1.g — Пример агрегации

Композиция — ещё один частный случай ассоциации, показывающий, что дочерние элементы входят в состав родительского элемента, но, в отличие от агрегации, здесь дочерние элементы не могут существовать самостоятельно (без родительского элемента).

В примере ниже (рисунок 4.1.h) показано, что текст входит в состав музыкального произведения (и сам по себе не существует, т.е. это именно «текст песни»). В примере с уточнёнными параметрами показано, что:

- каждый текст обязан относиться хотя бы к одному музыкальному произведению;
- текст может относиться к более, чем одному музыкальному произведению;
- к каждому музыкальному произведению может относиться от нуля до бесконечности текстов.



Самый общий вид Уточнение некоторых параметров

Рисунок 4.1.h — Пример композиции

Обобщение — вид связи, показывающий, что её дочерний элемент является частным случаем родительского.

Обобщение очень часто используется в программировании (класс-родитель и класс-потомок как частный случай класса-родителя), а в проектировании баз данных обычно сразу выражается в виде той или иной ассоциации, но на начальных этапах проектирования инфологического уровня обобщения имеют право на существование, т.к. могут отражать реальное положение дел в предметной области.

В примере ниже (рисунок 4.1.i) показано, что контрактор является частным случаем сотрудника. Также именно обобщение использовано в рисунке 4.1.e для иллюстрации иерархии UML-связей. Обратите внимание, что у обобщения не бывает мощности, т.к. выражения вида «сотрудник — это N контракторов» лишено практического смысла, здесь показана именно иерархия, которая по определению не имеет такого свойства как «мощность связи».

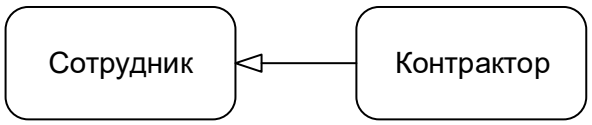


Рисунок 4.1.i — Пример обобщения

Реализация — вид связи, показывающий, что дочерний элемент реализует поведение, задаваемое родительским элементом.

Как и обобщение, реализация широко используется в программировании (интерфейс и реализующие его классы), а в проектировании баз данных она выражается в виде той или иной ассоциации, но, опять же, на начальных этапах проектирования инфологического уровня реализация может быть применена в UML-диаграмме для отражения особенностей предметной области.

В примере ниже (рисунок 4.1.j) показано, что техническое решение подчинено стандарту, т.к. обязано «вести себя» так, как сказано в стандарте.

Несмотря на то, что многие инструменты позволяют выставить «мощность связи реализации», в классическом варианте на UML-схемах она не указывается.

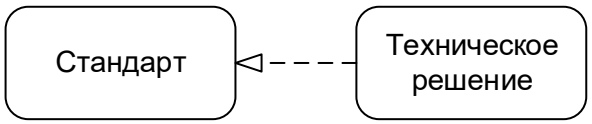
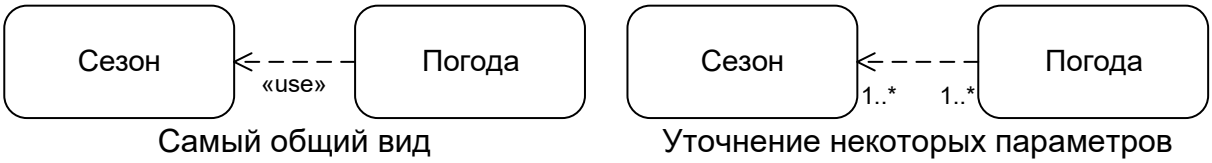


Рисунок 4.1.j — Пример реализации

Зависимость — вид связи, показывающий, что изменения в родительском элементе обязательно приводят к изменениям в дочернем элементе (но не наоборот).

В примере ниже (рисунок 4.1.k) показано, что изменение сезона должно приводить к изменению погоды (обратное — неверно). В примере с уточнёнными параметрами показано, что:

- у каждого сезона есть хотя бы один вид погоды (но может быть и больше, до бесконечности);
- каждый вид погоды обязан относиться хотя бы к одному сезону (но может относиться и к большему количеству сезонов, до бесконечности).



Самый общий вид — Уточнение некоторых параметров

Рисунок 4.1.k — Пример зависимости

Осталось рассмотреть, как UML-понятия «класс», «атрибут» и «метод» связаны с проектированием баз данных.

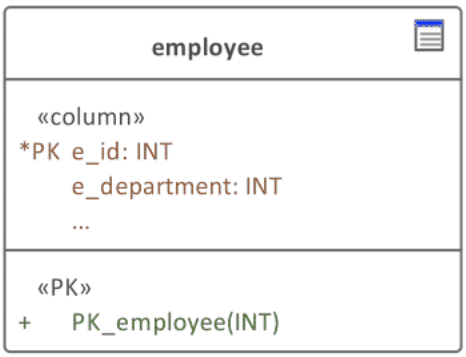


Рисунок 4.1.l — Связь UML-понятий «класс», «атрибут» и «метод» с проектированием баз данных

Здесь для иллюстрации будет достаточного одного рисунка 4.1.l — прямоугольник обозначает класс, в верхнем блоке приведено имя класса, в среднем — атрибуты класса, а в нижнем — методы класса.

Проще всего разобраться с самим «классом» (в проектировании баз данных это будет «схема отношения» или просто «отношение» в зависимости от контекста) и «атрибутом» (в проектировании баз данных это будет «атрибут отношения»).

С методами ситуация выглядит чуть необычнее, но она идеально отражает логику работы СУБД: к «методам отношения» будут относиться первичный ключ, индексы и иные *операции*.

И это не опечатка — это именно операции. С точки зрения человека, например, первичный ключ — это просто поле или несколько полей, обладающих определёнными свойствами, но с точки зрения СУБД — это ещё и необходимость *выполнять* проверку уникальности значений. Аналогично, индекс — это необходимость *выполнять* набор действий по его построению и актуализации. Потому это — именно *операции*, и они отражаются в блоке методов (т.е. того, что отношение «умеет делать»).

Можно смело утверждать, что такого краткого рассмотрения UML будет достаточно для понимания всех примеров, представленных в этой книге.

Существует очень много инструментов, используемых для проектирования моделей баз данных в UML-нотации, они интенсивно развиваются, и описание работы с ними почти всегда можно найти в прилагаемом руководстве пользователя.

Отметим лишь, что одним из наиболее популярных является Sparx Enterprise Architect, в котором и созданы все UML-диаграммы, представленные в данной книге.

Переходим к рассмотрению большого примера.

Задание 4.1.b: на основе инфологической схемы базы данных «Банк»{409} создайте соответствующую семантическую схему.

4.1.3. Пример проектирования на инфологическом уровне

До сих пор для иллюстрации тех или иных идей были использованы простые не связанные между собой примеры, в которых чаще всего были задействованы одна-две схемы отношения.

Но с этого момента мы приступаем к рассмотрению «долгоиграющего» и очень объёмного примера: мы будем проектировать полноценную базу данных — пусть и учебную, но максимально приближенную к реальности.

Итак, представим, что нам предстоит создать файлообменный сервис, т.е. веб-приложение, в котором пользователи могут размещать файлы для скачивания другими пользователями.

Осознанно до предела упростим[195] перечень и форму представления требований к приложению, получив в первом приближении такой результат:

1. Приложение содержит несколько страниц (их количество, иерархия, информационное наполнение может быть произвольным).
2. У приложения есть пользователи, которые могут создавать группы и объединяться в такие группы.
3. У каждого пользователя может быть несколько ролей, наделённых набором прав.
4. Пользователи могут закачивать и скачивать файлы, открывая к ним доступ для отдельных пользователей, для групп пользователей и «внешний» доступ.
5. Пользователи могут оставлять комментарии к представленным на сервера файлам.
6. У файла должен быть рейтинг.
7. Можно «отвечать на комментарии» (до глубины вложенности в 10 уровней).
8. Каждый файл обязательно должен относиться к одной из категорий файлов, определяющих перечень распространяемых на файл ограничений.
9. Приложение должно вести протокол всех действий всех пользователей.
10. Должна быть предусмотрена возможность блокировки пользователей, групп пользователей и незарегистрированных посетителей (по ip).
11. Приложение должно с минимальными временными задержками показывать текущую статистику по количеству пользователей, количеству и объёму файлов, количеству и объёму скачанных файлов.

Рассмотрим перечень сущностей, необходимость в которых проистекает из наличия таких требований:

№ тр.	Текст требования	Список сущностей
1	Приложение содержит несколько страниц (их количество, иерархия, информационное наполнение может быть произвольным)	Страница.
2	У приложения есть пользователи, которые могут создавать группы и объединяться в такие группы.	Пользователь. Группа пользователей.
3	У каждого пользователя может быть несколько ролей, наделённых набором прав.	Роль. Право.
4	Пользователи могут закачивать и скачивать файлы, открывая к ним доступ для отдельных пользователей, для групп пользователей и «внешний» доступ.	Файл.
5	Пользователи могут оставлять комментарии к представленным на сервера файлам.	Комментарий.
6	У файла должен быть рейтинг.	Оценка.

[195] См. раздел «2.2. Тестирование документации и требований» в книге «Тестирование программного обеспечения. Базовый курс». (С.С. Куликов) [https://svyatoslav.biz/software_testing_book/]

7	Можно «отвечать на комментарии» (до глубины вложенности в 10 уровней).	Комментарий.
8	Каждый файл обязательно должен относиться к одной из категорий файлов, определяющих перечень распространяемых на файл ограничений.	Категория файлов.
9	Приложение должно вести протокол всех действий всех пользователей.	Протокол.
10	Должна быть предусмотрена возможность блокировки пользователей, групп пользователей и незарегистрированных посетителей (по ip).	Причина блокировки.
11	Приложение должно с минимальными временными задержками показывать текущую статистику по количеству пользователей, количеству и объёму файлов, количеству и объёму скачанных файлов.	Статистика.

Итак, мы получили следующий перечень сущностей:

1. Страница.
2. Пользователь.
3. Группа пользователей.
4. Роль.
5. Право.
6. Файл.
7. Оценка.
8. Комментарий.
9. Категория файлов.
10. Протокол.
11. Причина блокировки.
12. Статистика.

Здесь пока не учтены «технические отношения» (для связей «многие ко многим»). Также здесь вполне могут появиться и иные сущности, которые пока не были учтены.

К сожалению, в формате книги невозможно передать процесс общения с заказчиком (фактически, это, чаще всего, обычная беседа в формате «вопрос-ответ»), но предположим, что после общения и прояснения подробностей мы получили следующую картину (добавились атрибуты сущностей, самих сущностей стало больше):

1. Страница:
    a. Родительская страница.
    b. Название страницы (для отображения на самой странице).
    c. Имя страницы в меню.
    d. Заглавие страницы (HTML-тег TITLE).
    e. Ключевые слова (HTML-тег meta … keywords).
    f. Описание страницы (HTML-тег meta … description).
    g. Текстовое наполнение страницы.
2. Пользователь:
    a. Логин.
    b. Пароль.
    c. E-mail.
    d. Дата и время регистрации.
    e. Дата рождения.
    f. Бонус по количеству закачанного.

3. Группа пользователей:
    a. Владелец (создатель) группы.
    b. Название группы.
    c. Описание группы.
4. Роль:
    a. Название роли.
    b. Ограничения по объёму закачиваемых файлов.
    c. Ограничения по объёму скачиваемых файлов.
    d. Ограничения по количеству закачиваемых файлов.
    e. Ограничения по количеству скачиваемых файлов.
    f. Ограничения по скорости скачивания.
5. Право:
    a. Название права.
    b. Описание права.
6. Файл:
    a. Размер (в байтах).
    b. Дата добавления.
    c. Срок хранения (до какой даты и какого времени).
    d. Исходное имя (без расширения).
    e. Исходное расширение.
    f. Имя на сервере.
    g. Контрольная сумма.
    h. Ссылка на удаление (для незарегистрированных пользователей).
7. Ссылка на «публичное» скачивание файла:
    a. К какому файлу.
    b. Значение ссылки.
    c. Срок действия ссылки (дата и время, после которых ссылка считается недействительной и должна быть удалена).
    d. Пароль на скачивание (если есть).
8. Параметр доступа к файлу:
    a. К какому файлу.
    b. Какое действие.
    c. Кому разрешено.
9. Оценка:
    a. Какой файл оценивается.
    b. Кто оценивает.
    c. Значение оценки.
10. Комментарий:
    a. К какому файлу.
    b. К какому комментарию.
    c. Текст комментария.
    d. Кто оставил комментарий.
    e. Дата и время добавления комментария.
11. Категория файлов:
    a. Название категории.
    b. Возрастные ограничения (если есть).
12. Возрастное ограничение:
    a. Минимальный возраст (в годах).
    b. Название ограничения.
    c. Описание ограничения.

13. Протокол:
    a. Пользователь.
    b. Ip-адрес.
    c. Операция.
    d. Файл (если задействован).
    e. Дата и время выполнения действия.
    f. Параметры действия.
14. Архив протокола (для записей старше месяца):
    a. Пользователь.
    b. Ip-адрес.
    c. Операция.
    d. Файл (если задействован).
    e. Дата и время выполнения действия.
    f. Параметры действия.
15. Причина блокировки:
    a. Название причины.
    b. Описание причины.
16. Чёрный список IP-адресов:
    a. IP-адрес в формате IPv4.
    b. IP-адрес в формате IPv6.
    c. Дата и время, до которых действительна блокировка.
    d. Причина блокировки.
17. Статистика:
    a. Всего зарегистрированных пользователей.
    b. Пользователей зарегистрировалось сегодня.
    c. Всего закачано файлов.
    d. Файлов закачано сегодня.
    e. Общий объём закачанных файлов.
    f. Объём файлов, закачанных сегодня.
    g. Всего скачано файлов.
    h. Файлов скачано сегодня.
    i. Общий объём скачанных файлов.
    j. Объём файлов, скачанных сегодня.

И снова предположим, что, обдумав и обсудив с коллегами полученный результат, мы через пару дней вернулись к обсуждению с заказчиком, в процессе которого постепенно начали прописывать технические детали.

Получилось следующее:

1. Страница:
    a. Идентификатор.
    b. Родительская страница (rFK).
    c. Название страницы (для отображения на самой странице).
    d. Имя страницы в меню (уникальное в рамках одного уровня меню).
    e. Заглавие страницы (HTML-тег TITLE).
    f. Ключевые слова (HTML-тег meta … keywords).
    g. Описание страницы (HTML-тег meta … description).
    h. Текстовое наполнение страницы.

2. Пользователь:
    a. Идентификатор.
    b. Логин (уникальный).
    c. Пароль (sha256-хэш).
    d. E-mail (уникальный).
    e. Дата и время регистрации (с точностью до секунды).
    f. Дата рождения (с точностью до дня).
    g. Бонус в виде скорости по количеству закачанного (FK).
    h. Срок действия бонуса в виде скорости по количеству закачанного (до какой даты и времени, с точностью до секунды).
    i. Бонус в виде объёма по количеству закачанного (FK).
    j. Срок действия бонуса в виде объёма по количеству закачанного (до какой даты и времени, с точностью до секунды).
3. Бонус:
    a. Идентификатор.
    b. За какое количество закачанных файлов выдаётся.
    c. За какой объём закачанных файлов выдаётся.
    d. Сколько добавляет к скорости скачивания.
    e. Сколько добавляет к объёму скачивания.
4. Группа пользователей:
    a. Идентификатор.
    b. Владелец (создатель) группы (FK).
    c. Название группы (уникальное).
    d. Описание группы.
5. Роль:
    a. Идентификатор.
    b. Название роли (уникальное).
    c. Ограничения по объёму закачиваемых файлов (в байтах).
    d. Ограничения по объёму скачиваемых файлов (в байтах).
    e. Ограничения по количеству закачиваемых файлов.
    f. Ограничения по количеству скачиваемых файлов.
    g. Ограничения по скорости скачивания (в байтах в секунду).
6. Право:
    a. Идентификатор.
    b. Название права (уникальное).
    c. Описание права.
7. Файл:
    a. Идентификатор.
    b. Размер (в байтах).
    c. Дата и время добавления (с точностью до секунды).
    d. Срок хранения (до какой даты и какого времени, с точностью до секунды).
    e. Исходное имя (без расширения).
    f. Исходное расширение.
    g. Имя на сервере (sha256-хэш).
    h. Контрольная сумма (sha256-хэш).
    i. Ссылка на удаление (для незарегистрированных пользователей, sha256-хэш).

8. Ссылка на «публичное» скачивание файла:
    a. Идентификатор.
    b. К какому файлу (FK).
    c. Значение ссылки (sha256-хэш).
    d. Срок действия ссылки (дата и время, после которых ссылка считается недействительной и должна быть удалена, с точностью до секунды).
    e. Пароль на скачивание (если есть, sha256-хэш).
9. Параметр доступа к файлу:
    a. Идентификатор.
    b. К какому файлу (FK).
    c. Какое действие (FK).
    d. Какому пользователю разрешено (FK).
    e. Какой группе разрешено (FK).
    f. Признак «разрешено всем зарегистрированным пользователям».
    g. Признак «разрешено вообще всем».
10. Оценка:
    a. Идентификатор.
    b. Какой файл оценивается (FK).
    c. Какой пользователь оценивает (FK).
    d. Значение оценки (от 1 до 10).
11. Комментарий:
    a. Идентификатор.
    b. К какому файлу (FK).
    c. К какому комментарию (FK).
    d. Текст комментария.
    e. Какой пользователь оставил комментарий (FK).
    f. Дата и время добавления комментария (с точностью до секунды).
12. Категория файлов:
    a. Идентификатор.
    b. Название категории (уникальное).
    c. Возрастные ограничения (если есть, FK).
13. Возрастное ограничение:
    a. Идентификатор.
    b. Минимальный возраст (в годах).
    c. Название ограничения (уникальное).
    d. Описание ограничения.
14. Протокол:
    a. Пользователь (FK, NULL для незарегистрированных).
    b. Ip-адрес.
    c. Операция (FK).
    d. Файл (если задействован, FK).
    e. Дата и время выполнения действия (с точностью до секунды).
    f. Параметры действия (если есть, текст).
15. Операция:
    a. Идентификатор.
    b. Название (уникальное).

16. Архив протокола (для записей старше месяца):
    a. Пользователь (FK, NULL для незарегистрированных).
    b. Ip-адрес.
    c. Операция (FK).
    d. Файл (если задействован, FK).
    e. Дата и время выполнения действия (с точностью до секунды).
    f. Параметры действия (если есть, текст).
17. Причина блокировки:
    a. Идентификатор.
    b. Название причины (уникальное).
    c. Описание причины.
18. Чёрный список IP-адресов:
    a. IP-адрес в формате IPv4 (значение уникально).
    b. IP-адрес в формате IPv6 (значение уникально).
    c. Дата и время, до которых действительна блокировка (с точностью до секунды).
    d. Причина блокировки (FK).
19. Статистика:
    a. Всего зарегистрированных пользователей.
    b. Пользователей зарегистрировалось сегодня.
    c. Всего закачано файлов.
    d. Файлов закачано сегодня.
    e. Общий объём закачанных файлов (в байтах).
    f. Объём файлов, закачанных сегодня (в байтах).
    g. Всего скачано файлов.
    h. Файлов скачано сегодня.
    i. Общий объём скачанных файлов (в байтах).
    j. Объём файлов, скачанных сегодня (в байтах).

Как вы можете сами легко убедиться, текстовое описание уже занимает почти три страницы, и здесь ведь ещё не отражены связи и промежуточные отношения для связей «многие ко многим». Мы пока не отказываемся от текстового представления, но дополним его графическим, чтобы продемонстрировать, насколько оно компактнее и нагляднее (см. рисунок 4.1.m).

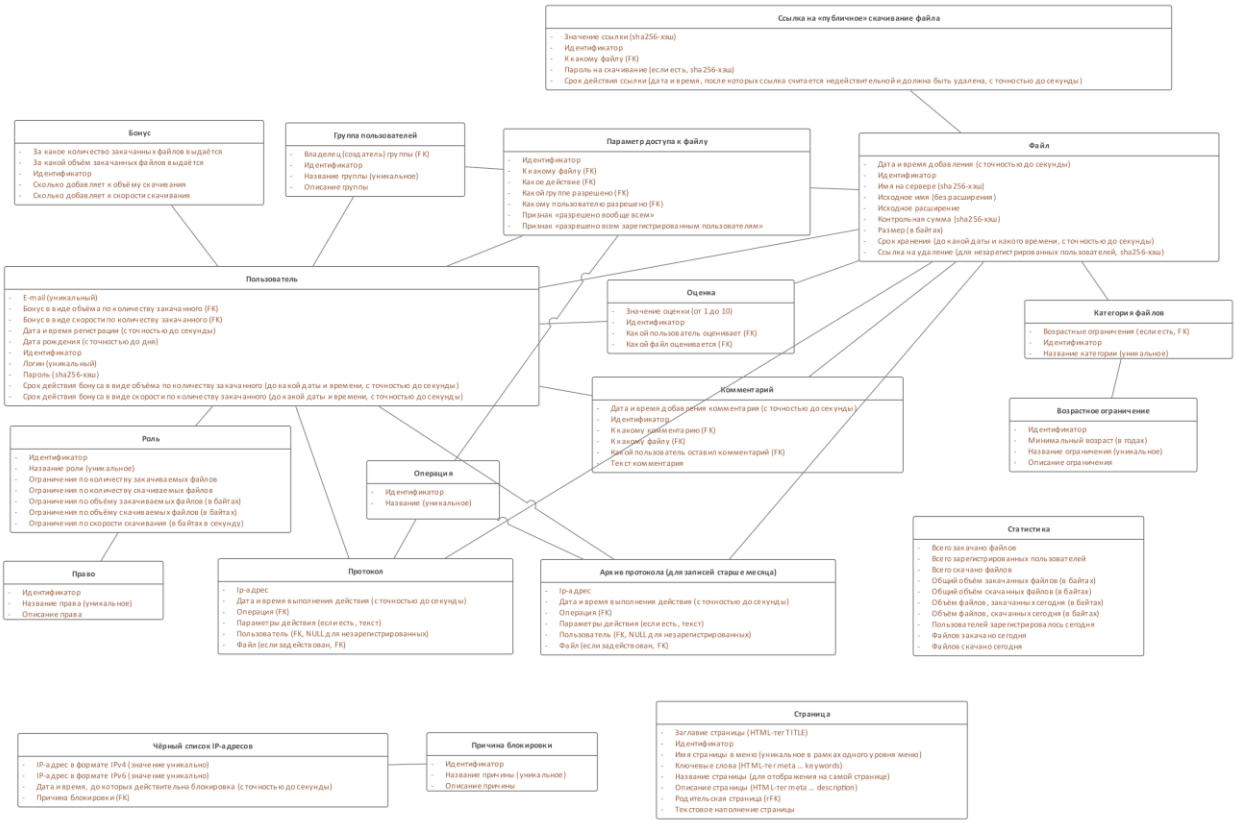


Рисунок 4.1.m — Графическое представление результатов проектирования на инфологическом уровне[196]

Вместо трёх страниц текста мы получили одну картинку (которая помещается на один экран), на которой не только отображена вся та же самая информация, но ещё и добавлены связи между сущностями, что даёт нам дополнительную информацию и повышает наглядность.

В данном конкретном примере была применена небольшая «хитрость»: пусть мы и находимся сейчас на инфологическом уровне, но и в текстовое описание, и в графическое представление мы уже добавили искусственные первичные ключи{41} (поля «Идентификатор») и внешние ключи{46} — что, говоря строго формально, относится к даталогическому уровню, но уже сейчас очень помогает с технической точки зрения.

Эта «хитрость» наглядно иллюстрирует, что разбиение на уровни моделирования во многом носит условный характер, а сами границы между уровнями являются очень размытыми{8}.

Также эта «хитрость» даёт нам ещё одно преимущество: если внимательно вчитаться в формулировки описаний атрибутов сущностей, можно заметить, что это — почти готовые комментарии к полям таблиц, которые мы будем проектировать уже очень скоро.

[196] Полноразмерную картинку вы можете найти в раздаточном материале{4} или по этой ссылке: https://svyatoslav.biz/relational_databases_book_download/pics/conceptual_level_graphical_representation.png.

Чем сложнее проект, сложнее его база данных и сложнее предметная область, тем больше времени необходимо провести в общении с заказчиком для того, чтобы максимально полно выявить все сущности, атрибуты, связи и особенности бизнес-процессов, которые окажут влияние на схему базы данных (если этого не сделать, очень высока вероятность нарушить такое ключевое свойство базы данных как адекватность предметной области{11}).

Однако всё же стоит помнить, что на этапе проектирования на инфологическом уровне всё равно не получится учесть все нюансы (и именно поэтому нам придётся несколько раз пройти весь процесс проектирования «сверху-вниз{10}» и «снизу вверх{11}»).

Потому сейчас мы можем смело переходить к следующему этапу — проектированию на даталогическом уровне.

Задание 4.1.c: доработайте инфологическую схему базы данных «Банк»{409} так, чтобы устранить все обнаруженные в ней недостатки (насколько это возможно без привлечения «гипотетического заказчика» для получения ответов на возникающие вопросы).

4.2. Проектирование на даталогическом уровне

4.2.1. Цели и задачи проектирования на даталогическом уровне

Перед прочтением материала данной главы стоит повторить основы моделирования баз данных{7}.

Как и было сказано в определении даталогического уровня моделирования{9}, основной его целью является детализация инфологической модели и превращение её в схему, на которой ранее выявленные сущности, атрибуты и связи оформляются согласно правилам моделирования для выбранного вида базы данных (часто даже с учётом конкретной СУБД).

Здесь не прекращается анализ предметной области (скорее всего, появятся новые вопросы, ответы на которые вынудят нас внести изменения в инфологическую модель), но особое внимание на данном уровне стоит уделить уже рассмотренным ранее требованиям, предъявляемым к любой базе данных{11}.

Поскольку именно на даталогическом уровне формируется будущая структура базы данных, все принятые здесь решения (и, увы, допущенные ошибки) будут очень сильно влиять на степень адекватности базы данных предметной области, на удобство использования базы данных, на её производительность и защищённость её данных.

Фактически, вся информация, представленная ранее в разделах 2{20} и 3{162} данной книги, в полной мере относится к особенностям проектирования на даталогическом уровне.

Какие таблицы{20} создать? Почему именно такие? Может быть, есть лучший вариант? Какие у этих таблиц будут поля{20} (а также какие у этих полей будут типы данных и иные свойства)? Какие ключи{34} использовать? Какие связи{56} и как именно устанавливать? Может быть, уже видна часть индексов{107} и представлений{341}? Достаточно ли наша схема базы данных нормализована{212}? Не подвержена ли она каким-то аномалиям{162}? На все эти и многие другие вопросы придётся искать ответы в процессе проектирования базы данных на даталогическом уровне.

Если свести вышесказанное буквально к двум фразам, то получится:

- цель проектирования на даталогическом уровне — создать структуру будущей базы данных;
- задача проектирования на даталогическом уровне — сделать создаваемую структуру максимально качественной, избавленной от обнаружимых на данном этапе проблем.

Предвидеть и заранее устранить все проблемы всё равно не получится — и это совершенно нормально. Но в наших силах — хотя бы избавиться от наиболее очевидных, типичных ошибок.

Достичь этого во многом помогает т.н. «восходящее проектирование{11}»: мы уже видим структуру базы данных и уже можем анализировать, к каким результатам приведёт выполнение тех или иных запросов на такой структуре. И пока «дела не зашли слишком далеко», мы можем легко изменить структуру базы данных, если видим, что она обладает тем или иным недостатком.

И кроме представленной в разделах 2{20} и 3{162} данной книги общей информации, мы можем применять различные специфические техники и инструменты, которые могут сильно упростить нашу работу.

Рассмотрим их.

Задание 4.2.а: сформулируйте список вопросов, ответы на которые помогли бы вам улучшить даталогическую схему базы данных «Банк»{409}.

4.2.2. Инструменты и техники проектирования на даталогическом уровне

Прежде, чем приступить к рассмотрению непосредственно инструментов и техник, сделаем небольшое отступление и подчеркнём, что необходимо знать и уметь для того, чтобы успешно проектировать базы данных на даталогическом уровне.

В идеале	Как минимум
Глубоко понимать реляционную теорию.	Иметь представление о реляционной теории.
Знать «на уровне инстинктов» теорию нормализации.	Понимать хотя бы 1–3 нормальные формы.
Глубоко знать SQL.	Знать основы SQL.
Знать целевую СУБД на уровне, позволяющем проводить её тонкое администрирование.	Уметь устанавливать и настраивать целевую СУБД.
Уметь использовать средства проектирования.	Иметь много терпения для изучения средств проектирования.

Реляционная теория и теория нормализации в достаточной мере представлены в предыдущих разделах данной книги, для глубокого погружения в SQL вы можете обратиться к книге «Работа с MySQL, MS SQL Server и Oracle в примерах»[197].

Сложнее всего будет с последними двумя пунктами — и СУБД, и средства проектирования развиваются слишком стремительно, чтобы по ним существовали некие «фундаментальные» книги (они будут устаревать уже к моменту публикации), потому самое надёжное решение — читать официальную документацию по конкретной версии конкретной СУБД и конкретного средства проектирования, которые вы используете. Возможно, официальная документация и не будет написана предельно упрощённым языком, но она будет как минимум актуальной — и это преимущество в данном случае перевешивает все недостатки.

Вернёмся к теме данной главы.

Проектирование на инфологическом уровне предполагало интенсивное обсуждение формируемой модели с заказчиком, и потому там мы старались применять такие средства, которые были бы доступны и понятны «не техническому» человеку (который, например, может быть глубочайшим экспертом в международной логистике или иной предметной области, но совершенно не обязан понимать технические тонкости работы баз данных).

Начиная с даталогического уровня, мы уже в куда большей степени ориентируемся на технических специалистов, поэтому используемые здесь решения могут быть непонятны заказчику — и это нормально, т.к. мы будем вносить необходимые правки в инфологическую модель и продолжать говорить с заказчиком «на его языке», не перегружая его техническими подробностями.

И первое, что стоит сделать техническим специалистам — это прийти к нескольким ключевым соглашениям, которые в общем случае можно поделить на две группы:

- Соглашения относительно СУБД.
- Соглашения относительно БД.

Отсутствие или недостаточная проработанность подобных соглашений очень сильно снижает такое свойство базы данных как удобство использования{13}.

[197] «Работа с MySQL, MS SQL Server и Oracle в примерах» (С.С. Куликов) [https://svyatoslav.biz/database_book/]

Соглашения относительно СУБД могут включать в себя, например, следующие пункты (конкретные примеры будут в следующем разделе{308}):

- Вид СУБД.
- Конкретный продукт (конкретная СУБД).
- Минимальная поддерживаемая версия выбранной конкретной СУБД.
- Инфраструктурные особенности выбранной конкретной СУБД.

В этом списке пункты приведены в порядке убываниях их «судьбоносности» (так, например, переход в середине проекта с реляционной СУБД на иерархическую фактически будет означать начало проекта с нуля; а вот перенос нескольких серверов в облако может пройти почти безболезненно).

Поскольку выбор конкретной СУБД и её версии очень сильно влияет на дальнейшие технические решения, с этими вопросами тоже стоит разобраться до того, как на даталогическом уровне будет сделано много работы. Если придётся переделывать модель под другую версию выбранной СУБД или даже под другую СУБД, гарантированно потребуется очень много рутинной работы, а также, возможно, и интеллектуальной — если придётся искать новые технологические решения взамен уже созданным.

Соглашения относительно БД принимать проще, т.к. они слабо зависят от внешних факторов, а потому, как правило, почти не меняются. Что, однако, не снижает их важности. К таким решениям относятся (конкретные примеры будут в следующем разделе{308}):

- Соглашения об именовании структур.
- Соглашения об оформлении SQL-кода.
- Соглашения о комментариях.
- Иные специфические соглашения, зависящие от проекта (например, соглашения об API[198] в виде представлений{341} и хранимых подпрограмм{364}).

Конечно, никто не застрахован от таких изменений проектной ситуации, которые вынудят нас пересмотреть любое из только что обозначенных соглашений, но в наших силах продумать эти соглашения настолько хорошо, чтобы в будущем их не пришлось менять по нашей же собственной инициативе.

С инструментами на даталогическом уровне дела обстоят неоднозначно.

Для инфологического уровня нам в общем случае было достаточно любого текстового редактора (или любого удобного нам и представителям заказчика специализированного редактора UML (или иного графического языка) — таких инструментов много, и большинство специалистов хорошо умеет ими пользоваться).

Для даталогического уровня нам нужны специализированные инструменты, позволяющие оптимальным образом формировать структуру базы данных, многократно её пересматривать и анализировать, а в конечном итоге и экспортировать в полноценный SQL-код для импорта в СУБД.

Каждый производитель СУБД традиционно предоставляет свои собственные инструменты, например:

- MySQL Workbench[199] для MySQL.
- SQL Server Management Studio[200] для MS SQL Server.
- SQL Developer Data Modeler[201] для Oracle.
- И так далее — таких инструментов очень много.

---

[198] **API** (application programming interface) — a computing interface which defines interactions between multiple software intermediaries. («Wikipedia») [https://en.wikipedia.org/wiki/Application_programming_interface]

[199] «MySQL Workbench» [https://www.mysql.com/products/workbench/]

[200] «SQL Server Management Studio» [https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms]

[201] «SQL Developer Data Modeler» [https://www.oracle.com/database/technologies/appdev/datamodeler.html]

Это, безусловно, очень мощные инструменты, созданные с учётом всех особенностей целевой СУБД, и они однозначно заслуживают внимания. Если бы не одно «но»: они больше пригодятся вам для проектирования на следующем, физическом уровне{315}.

У каждого из перечисленных инструментов долгая история развития, их создатели ставили во главу угла разные цели, и в итоге каждый такой инструмент хорош в чём-то своём, но едва ли может претендовать на звание «эталонного решения».

На такое звание могут претендовать инструменты, изначально созданные для проектирования «баз данных в принципе», например:

- Sparx Systems Enterprise Architect[202] (большинство иллюстраций со схемами баз данных в данной книге сделаны именно с использованием этого инструмента).
- DbSchema[203].
- DbDiagram.io[204].
- И так далее — таких инструментов очень много.

Почему эти «универсальные» инструменты выигрывают? Вот лишь несколько самых очевидных преимуществ:

- Ни один из них не работает напрямую с физической базой данных (что сводит к нулю ваши шансы однажды уничтожить уже работающую базу данных заказчика).
- Эти инструменты поддерживают синтаксис и особенности многих СУБД, что позволяет вам сравнивать принимаемые решения для разных СУБД, переходить (конвертировать модели) с одной СУБД на другую или даже работать в таких необычных ситуациях, когда часть базы данных работает под управлением одной СУБД, а часть — под управлением другой СУБД.
- Создатели этих инструментов учли многие недостатки «узкоспециализированных» аналогов и постарались их исправить.
- Такие инструменты, как правило, обладают гораздо более дружественным интерфейсом, более развитыми средствами совместной (командной) работы и многими иными конкурентными преимуществами.

Естественно, вы имеете полное право выбора как между этими группами инструментов, так и между конкретными продуктами. А на стадии обучения рекомендуется попробовать поработать с хотя бы 3–5 принципиально различными инструментами, чтобы на личном опыте понять их различия и сформировать собственные предпочтения.

Задание 4.2.b: создайте даталогическую схему базы данных «Банк»{409} с использованием MySQL Workbench.

[202] «Sparx Systems Enterprise Architect» [https://sparxsystems.com/products/ea/index.html]
[203] «DbSchema» [https://dbschema.com]
[204] «DbDiagram.io» [https://dbdiagram.io]

4.2.3. Пример проектирования на даталогическом уровне

Как и было отмечено ранее, приступая к проектированию БД на даталогическом уровне, стоит принять ряд соглашений{305}. Сделаем это.

Соглашения относительно СУБД:

- Вид СУБД: реляционная.
- Конкретный продукт (конкретная СУБД): MySQL.
- Минимальная поддерживаемая версия выбранной конкретной СУБД: 8.0 и новее (т.к. более ранние версии имеют ряд технических ограничений).
- Инфраструктурные особенности выбранной конкретной СУБД: отдельный сервер[205].

Соглашения относительно БД:

- Соглашения об именовании структур:
  - При именовании таблиц и полей используется только нижний регистр.
  - Слова в именах таблиц и полей отделены друг от друга символом «_».
  - В именах таблиц существительные используются в единственном числе (например, «file», а не «files»). В именах полей множественное число допускается, но не рекомендуется.
  - Имена полей начинаются с префиксов из первых букв имён таблицы.
  - Имена ограничений начинаются с префикса «UNQ_» и содержат имена всех входящий в ограничение полей.
  - Имена триггеров начинаются с префикса «TRG_» и содержат в себе имя таблицы и название момента срабатывания.
- Соглашения об оформлении SQL-кода:
  - Все ключевые слова языка SQL пишутся в верхнем регистре.
  - Все имена структур БД обязательно должны быть заключены в обратные апострофы, т.е. символы «`».
- Соглашения о комментариях:
  - Комментарии оформляются для всех структур БД.
- Иные специфические соглашения, зависящие от проекта:
  - Все поля, содержащие дату и время, имеют тип **`INTEGER`** и хранят UNIXTIME-значения.
  - Все поля, содержащие только дату, имеют тип **`DATE`**.
  - Все первичные ключи — искусственные, автоинкрементируемые, беззнаковые.

В качестве основного инструмента проектирования мы будем использовать Sparx Enterprise Architect{307}, однако подчеркнём, что в особо сложных случаях и/или при проектировании большой БД может существовать промежуточный этап, на котором мы по-прежнему можем опираться на текстовое описание модели.

В случае с нашим конкретным учебным проектом явной необходимости в таком промежуточном этапе нет, но для полноты картины приведём соответствующий пример. Как правило, здесь будет использоваться табличное представление, т.к. для восприятия структурированной информации оно оказывается удобнее списков.

В процессе формирования этого примера мы также внесём некоторые правки в модель, уточнив то, что мы «забыли» на инфологическом уровне (см. задание 4.2.d{314}).

[205] В реальности, скорее всего, мы бы сразу ориентировались на кластер серверов, т.к. подобный проект предполагает достаточно высокую нагрузку, с которой отдельный сервер едва ли справится. Но из соображений упрощения учебного материала остановимся на более тривиальном решении.

Таблица БД	Поле таблицы БД	Тип данных	Комментарии
Страница	Идентификатор	SMALLINT	Первичный ключ
	Родительская страница (rFK)	SMALLINT	
	Название страницы (для отображения на самой странице)	VARCHAR(500)	
	Имя страницы в меню (уникальное в рамках одного уровня меню)	VARCHAR(100)	Создать контролирующий триггер
	Заглавие страницы (HTML-тег TITLE)	VARCHAR(500)	
	Ключевые слова (HTML-тег meta … keywords)	VARCHAR(500)	
	Описание страницы (HTML-тег meta … description)	VARCHAR(500)	
	Текстовое наполнение страницы	TEXT	
Пользователь	Идентификатор	BIGINT	Первичный ключ
	Логин (уникальный)	VARCHAR(100)	Уникальное значение
	Пароль (sha256-хэш)	CHAR(64)	
	E-mail (уникальный)	VARCHAR(150)	Уникальное значение
	Дата и время регистрации (с точностью до секунды)	INT	
	Дата рождения (с точностью до дня)	DATE	
	Бонус в виде скорости по количеству закачанного (FK)	SMALLINT	
	Срок действия бонуса в виде скорости по количеству закачанного (до какой даты и времени, с точностью до секунды)	INT	
	Бонус в виде объёма по количеству закачанного (FK)	SMALLINT	
	Срок действия бонуса в виде объёма по количеству закачанного (до какой даты и времени, с точностью до секунды)	INT	
	Сколько файлов пользователь закачал	BIGINT	Агрегирующее поле, обновляется триггером
	Сколько файлов пользователь скачал	BIGINT	Агрегирующее поле, обновляется триггером
	Сколько комментариев оставил пользователь	BIGINT	Агрегирующее поле, обновляется триггером
	Сколько оценок оставил пользователь	BIGINT	Агрегирующее поле, обновляется триггером
	Причина блокировки (FK)	SMALLINT	
	До какой даты-времени (с точностью до секунды) действует блокировка	INT	
Бонус	Идентификатор	SMALLINT	Первичный ключ
	За какое количество закачанных файлов выдаётся	BIGINT	
	За какой объём закачанных файлов (в байтах) выдаётся	BIGINT	
	Сколько добавляет к скорости скачивания (байт в секунду)	BIGINT	
	Сколько добавляет к объёму скачивания (в байтах)	BIGINT	

Таблица БД	Поле таблицы БД	Тип данных	Комментарии
Группа пользователей	Идентификатор	SMALLINT	Первичный ключ
	Владелец (создатель) группы (FK)	BIGINT	
	Название группы (уникальное)	VARCHAR(100)	Уникальное значение
	Описание группы	TEXT	
Роль	Идентификатор	SMALLINT	Первичный ключ
	Название роли (уникальное)	VARCHAR(100)	Уникальное значение
	Ограничения по объёму закачиваемых файлов (в байтах)	BIGINT	NULL, если ограничения нет
	Ограничения по объёму скачиваемых файлов (в байтах)	BIGINT	NULL, если ограничения нет
	Ограничения по количеству закачиваемых файлов	BIGINT	NULL, если ограничения нет
	Ограничения по количеству скачиваемых файлов	BIGINT	NULL, если ограничения нет
	Ограничения по скорости закачивания (в байтах в секунду)	BIGINT	NULL, если ограничения нет
	Ограничения по скорости скачивания (в байтах в секунду)	BIGINT	NULL, если ограничения нет
Право	Идентификатор	SMALLINT	Первичный ключ
	Название права (уникальное)	VARCHAR(100)	Уникальное значение
	Описание права	TEXT	
Файл	Идентификатор	BIGINT	Первичный ключ
	Владелец	BIGINT	
	Размер (в байтах)	BIGINT	
	Дата и время добавления (с точностью до секунды)	INT	
	Срок хранения (до какой даты и какого времени, с точностью до секунды)	INT	NULL, если хранить бессрочно
	Исходное имя (без расширения)	VARCHAR(1000)	
	Исходное расширение	VARCHAR(1000)	
	Имя на сервере (sha256-хэш)	CHAR(64)	Уникальное значение
	Контрольная сумма (sha256-хэш)	CHAR(64)	
	Ссылка на удаление (для незарегистрированных пользователей, sha256-хэш)	CHAR(64)	Уникальное значение
Ссылка на «публичное» скачивание файла	Идентификатор	BIGINT	Первичный ключ
	К какому файлу (FK)	BIGINT	
	Значение ссылки (sha256-хэш)	CHAR(64)	Уникальное значение
	Срок действия ссылки (дата и время, после которых ссылка считается недействительной и должна быть удалена, с точностью до секунды)	INT	NULL, если ссылка бессрочная
	Пароль на скачивание (если есть, sha256-хэш)	CHAR(64)	NULL, если пароль не требуется
Параметр доступа к файлу	Идентификатор	BIGINT	Первичный ключ
	К какому файлу (FK)	BIGINT	
	Какое действие (FK)	SMALLINT	

Таблица БД	Поле таблицы БД	Тип данных	Комментарии
	Какому пользователю разрешено (FK)	BIGINT	Из этих двух полей одно обязательно должно быть NULL, другое — не NULL
	Какой группе разрешено (FK)	SMALLINT	
	Признак «разрешено всем зарегистрированным пользователям»	BIT(1)	
	Признак «разрешено вообще всем»	BIT(1)	
Оценка	Идентификатор	BIGINT	Первичный ключ
	Какой файл оценивается (FK)	BIGINT	
	Какой пользователь оценивает (FK)	BIGINT	
	Значение оценки (от 1 до 10)	TINYINT	
Комментарий	Идентификатор	BIGINT	Первичный ключ
	К какому файлу (FK)	BIGINT	Из этих двух полей одно обязательно должно быть NULL, другое — не NULL
	К какому комментарию (FK)	BIGINT	
	Текст комментария	TEXT	
	Какой пользователь оставил комментарий (FK)	BIGINT	
	Дата и время добавления комментария (с точностью до секунды)	INT	
	Выставленная оценка (FK)	BIGINT	Комментируя файл, можно не только писать текст, но и выставлять файлу оценку, оценку можно ставить только в комментариях верхнего уровня
Категория файлов	Идентификатор	SMALLINT	Первичный ключ
	Название категории (уникальное)	VARCHAR(100)	Уникальное значение
	Возрастные ограничения (если есть, FK)	SMALLINT	
Возрастное ограничение	Идентификатор	SMALLINT	Первичный ключ
	Минимальный возраст (в годах)	TINYINT	
	Название ограничения (уникальное)	VARCHAR(100)	Уникальное значение
	Описание ограничения	TEXT	
Протокол	Пользователь (FK, NULL для незарегистрированных)	BIGINT	
	Ip-адрес	VARCHAR(45)	
	Операция (FK)	SMALLINT	
	Файл (если задействован, FK)	BIGINT	NULL, если операция не связана с файлами
	Дата и время выполнения действия (с точностью до секунды)	INT	
	Параметры действия (если есть)	TEXT	Сериализованный массив
Операция	Идентификатор	SMALLINT	Первичный ключ
	Название (уникальное)	VARCHAR(100)	Уникальное значение
Архив протокола (для записей старше месяца)	Пользователь (FK, NULL для незарегистрированных).	BIGINT	
	Ip-адрес	VARCHAR(45)	
	Операция (FK)	SMALLINT	
	Файл (если задействован, FK)	BIGINT	NULL, если операция не связана с файлами

Таблица БД	Поле таблицы БД	Тип данных	Комментарии
	Дата и время выполнения действия (с точностью до секунды)	INT	
	Параметры действия (если есть)	TEXT	Сериализованный массив
Причина блокировки	Идентификатор	SMALLINT	Первичный ключ
	Название причины (уникальное)	VARCHAR(100)	Уникальное значение
	Описание причины	TEXT	
Чёрный список IP-адресов	IP-адрес в формате IPv4 (значение уникально)	CHAR(15)	Эти два поля не могут одновременно быть NULL (сделать триггер для проверки)
	IP-адрес в формате IPv6 (значение уникально)	CHAR(45)	
	Дата и время, до которых действительна блокировка (с точностью до секунды)	INT	
	Причина блокировки (FK)	SMALLINT	
Статистика	Всего зарегистрированных пользователей	BIGINT	Все поля этой таблицы являются агрегирующими и обновляются триггерами на соответствующих таблицах
	Пользователей зарегистрировалось сегодня	BIGINT	
	Всего закачано файлов	BIGINT	
	Файлов закачано сегодня	BIGINT	
	Общий объём закачанных файлов (в байтах)	BIGINT	
	Объём файлов, закачанных сегодня (в байтах)	BIGINT	
	Всего скачано файлов	BIGINT	
	Файлов скачано сегодня	BIGINT	
	Общий объём скачанных файлов (в байтах)	BIGINT	
	Объём файлов, скачанных сегодня (в байтах)	BIGINT	

Несмотря на то, что вышеприведённая таблица не очень сильно отличается от рассмотренного на инфологическом уровне списка сущностей и атрибутов{298}, она всё же содержит ряд важных дополнительных подробностей и может стать хорошим дополнением к проектной документации.

Теперь представим получившуюся модель графически (см. рисунок 4.2.а).

Здесь уже более двух десятков таблиц и большое количество связей между ними, да — модель получилась достаточно объёмной, а потому «на одной картинке» она сложно читается, но всё равно выглядит намного компактнее в сравнении с четырьмя страницами текстового описания. Особенно — если открыть исходный файл (см. раздаточный материал к книге{4}) в Sparx Enterprise Architect и выбрать удобный лично вам масштаб отображения.

Можно заметить, что на рисунке 4.2.а таблиц больше, чем было представлено ранее в текстовом описании{309}. Это связано с тем, что в процессе формирования модели были добавлены таблицы для связей «многие ко многим{59}», а также внесены иные небольшие правки (в для т.ч. адаптации к требованию удобства использования БД{13}).

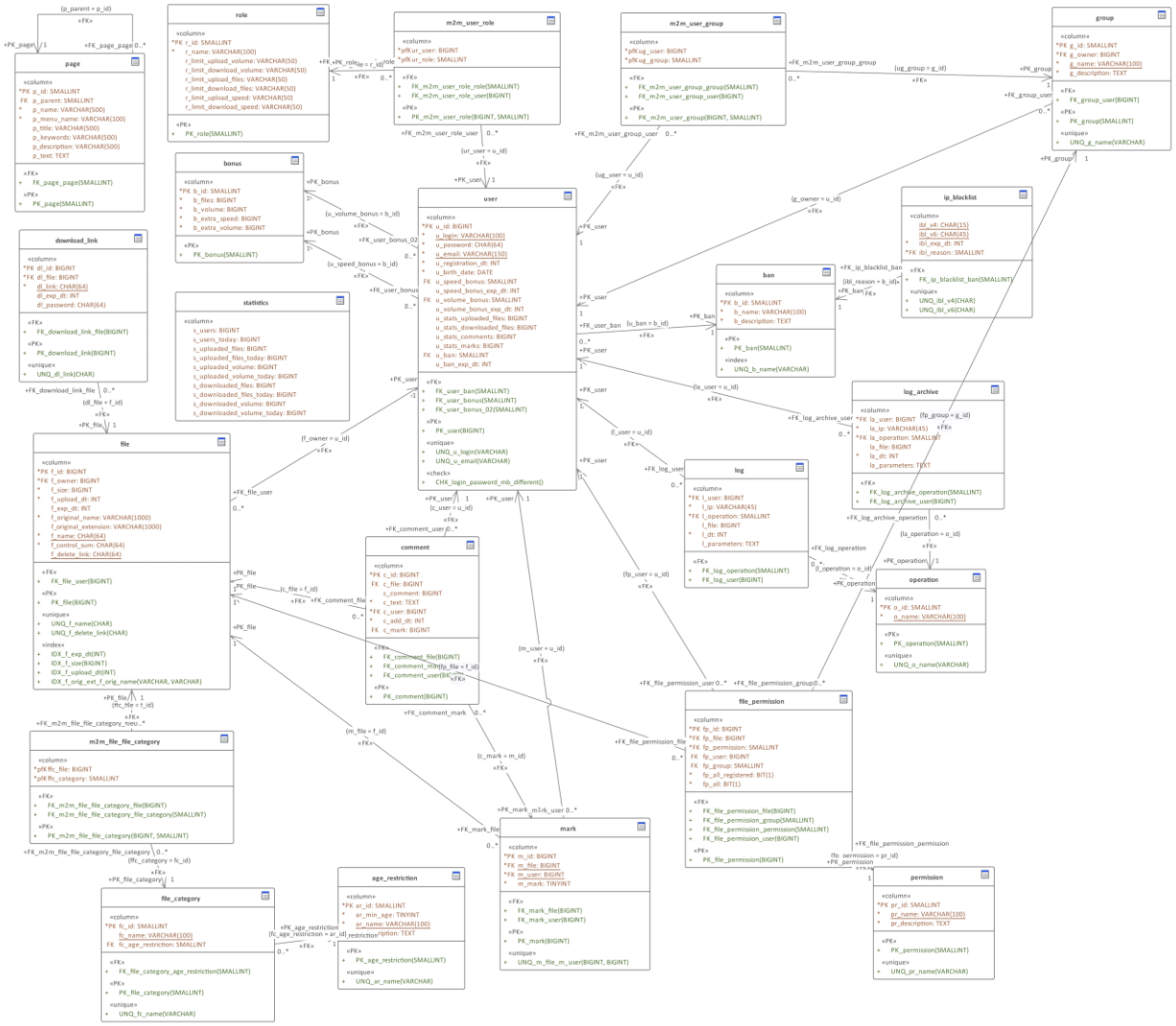


Рисунок 4.2.a — Графическое представление результатов проектирования на даталогическом уровне[206]

Также в процессе создания этой модели были добавлены почти все необходимые ограничения уникальности{110}, проведены связи между таблицами, и все комментарии были «вписаны» в модель (для того, чтобы в будущем при генерации SQL-кода они были перенесены в реальную базу данных).

Уже начиная с этого момента, мы можем (и даже должны) периодически генерировать SQL-код и импортировать получившийся результат в СУБД, чтобы избежать различных досадных ошибок, совершать которые склонен любой человек (если с моделью «что-то не так», импорт завершится неудачей, и СУБД подробно опишет причину в сообщениях об ошибках). Намного проще исправить подобные недочёты сейчас, пока модель ещё достаточно «сырая», и её доработка не потребует больших усилий.

[206] Полноразмерную картинку вы можете найти в раздаточном материале{4} или по этой ссылке: https://svyatoslav.biz/relational_databases_book_download/pics/logical_level_graphical_representation.png.

Так, например, при подготовке материала данной книги первая попытка переноса модели в СУБД завершилась следующей ошибкой:

MySQL | Сообщение СУБД об ошибке выполнения SQL-скрипта

```
1   ALTER TABLE `page` ADD CONSTRAINT `fk_page_page` FOREIGN KEY (`p_parent`)
2   REFERENCES `page` (`p_id`)
3   ON DELETE RESTRICT
4   ON UPDATE RESTRICT
5   Error Code: 1825. Failed to add the foreign key constraint on table 'page'.
6   Incorrect options in FOREIGN KEY constraint 'FK_page_page'.
```

И действительно, первичный ключ таблицы **`page`** был беззнаковым числом, в то время как рекурсивный внешний ключ этой же таблицы был обычным числом (способным принимать отрицательные значения).

После исправления этой (и пары аналогичных ей) ошибок импорт в СУБД прошёл успешно.

И тем не менее, до завершения проектирования остаётся ещё один очень насыщенный этап — создание модели базы данных на физическом уровне.

Задание 4.2.c: доработайте даталогическую схему базы данных «Банк»{409} так, чтобы устранить все обнаруженные в ней недостатки (насколько это возможно без привлечения «гипотетического заказчика» для получения ответов на возникающие вопросы).

Задание 4.2.d: ранее{308} в данной главе было отмечено, что в процессе промежуточного этапа проектирования (на котором мы по-прежнему опирались на текстовое описание модели) мы вносили некоторые правки, которые «забыли» на инфологическом уровне. Какие правки мы сделали? Составьте полный список.

Задание 4.2.e: какие ещё данные вы бы добавили в итоговую таблицу{309}, отражающую итог даталогического проектирования базы данных файлообменного сервиса? Внесите соответствующие правки.

4.3. Проектирование на физическом уровне

4.3.1. Цели и задачи проектирования на физическом уровне

Перед прочтением материала данной главы стоит повторить основы моделирования баз данных{7}.

Начиная с этого момента, мы вступаем в область, в которой очень мало универсальных решений, поскольку, как следует из определения физического уровня моделирования{10}, мы будем вынуждены максимально учитывать технические особенности и возможности конкретной СУБД.

И всё же некоторые общие цели и задачи можно выделить даже здесь, т.к. особый интерес для нас представляют:

- права доступа;
- кодировки;
- методы доступа;
- индексы;
- настройки СУБД.

Права доступа

В относительно небольших и простых проектах предполагается, что взаимодействие с СУБД идёт от имени одного пользователя — работающее с СУБД приложение всегда авторизуется с использованием одной и той же пары «логин-пароль», и само контролирует права своих пользователей (см. рисунок 4.3.а).

Рисунок 4.3.а — Работа с СУБД от имени одного пользователя

Этот вариант получил широкое распространение из-за своей простоты и скорости реализации. Но если мы рассмотрим ситуацию, когда с одной и той же базой данных может работать много разных приложений, а к безопасности предъявляются повышенные требования, имеет смысл переложить аутентификацию и контроль прав доступа на СУБД (см. рисунок 4.3.b).

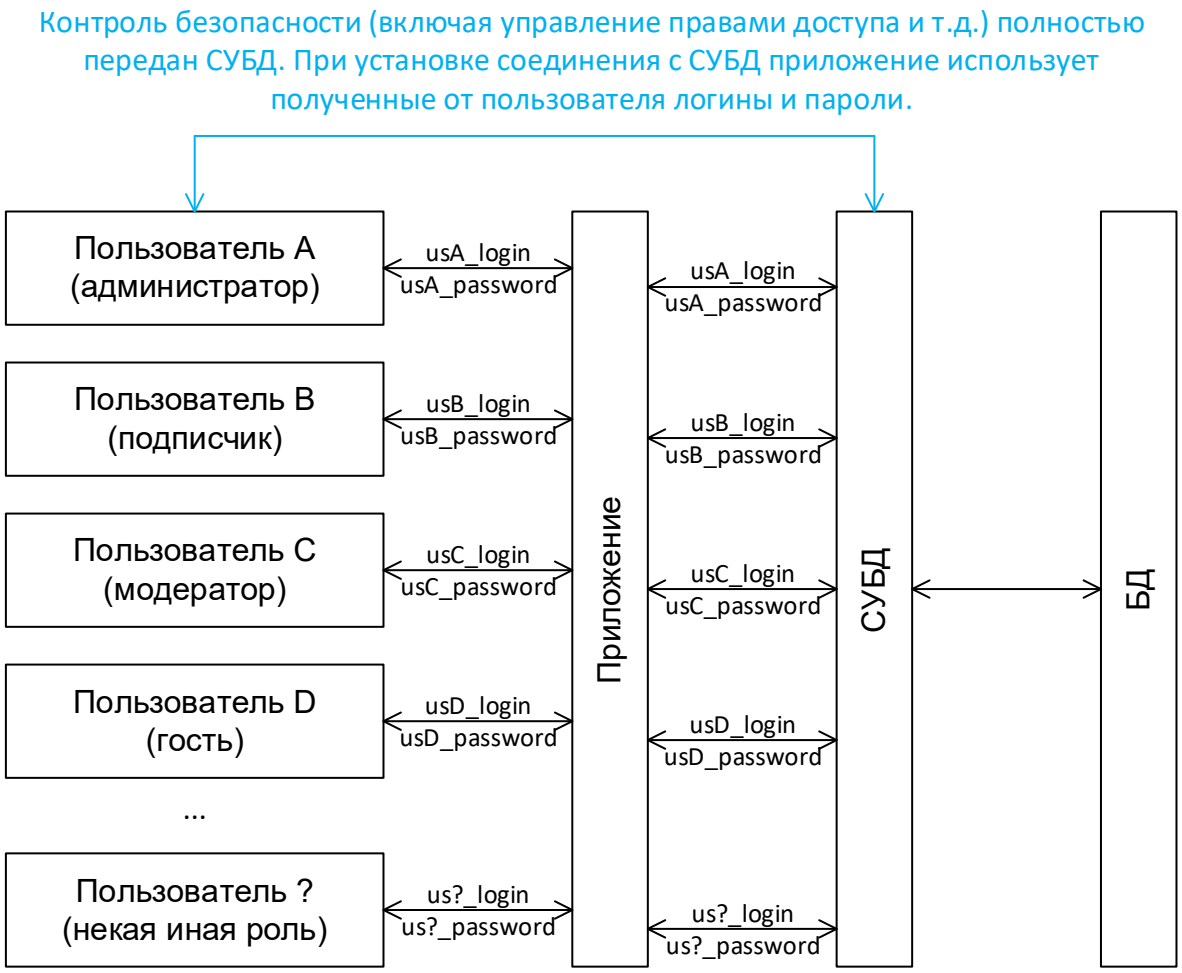


Рисунок 4.3.b — Работа с СУБД от имени нескольких пользователей

Да, такой подход намного более сложен в реализации, он требует не только постоянного администрирования СУБД (как минимум в плане управления имеющимися пользователями и их правами), но также требует серьёзных доработок в самой базе данных (например, для доступа к данным может быть создан дополнительный уровень абстракции в виде представлений{341} и хранимых процедур{364} — такой подход позволяет намного более гибко управлять правами доступа и защищать данные).

Но несмотря на свою ощутимо большую сложность, такой подход позволяет:

- обеспечить намного более высокий уровень безопасности;
- исключить необходимость дублирования модели безопасности во множестве отдельных приложений (модель один раз реализуется на уровне БД/СУБД и распространяется на всех клиентов);
- исключить ситуации осознанного или случайного «обхода» системы безопасности (когда, например, работа идёт напрямую с базой данных в процессе технического обслуживания или устранения неполадок).

Кодировки

Одной из самых частых ошибок начинающих при проектировании баз данных является недостаточное внимание кодировкам символов.

Ситуация многократно усугубляется тем, что русскоговорящие разработчики часто используют русифицированное программное обеспечение[207] (включая операционную систему), что приводит к автоматической конфигурации СУБД таким образом, что на компьютере у разработчика «всё работает». А при переносе на полноценный сервер — перестаёт.

Если при проектировании СУБД не указать явным образом кодировки базы данных, таблиц (а в некоторых СУБД — даже отдельных полей таблиц), то при переносе модели базы данных в СУБД все кодировки будут выставлены «по умолчанию», т.е. взяты из настроек той СУБД, в которую импортируется база данных.

Продемонстрируем на простом примере, к чему это приводит.

Предположим, что мы используем MySQL, и у разработчика на его компьютере кодировкой по умолчанию является `utf8`. При этом на одном из серверов, на которых будет работать создаваемая база данных, кодировкой по умолчанию является `latin1`.

Для наглядности запредельно упростим пример и создадим всего лишь одну таблицу с одним текстовым полем:

MySQL — Ошибочное создание таблицы (не указаны кодировки)

```
1    CREATE TABLE `words`
2    (
3     `word` VARCHAR(255) NULL
4    )
```

Наполним эту таблицу данными (вернее — безуспешно попытаемся):

MySQL — Ошибка при добавлении русского текста в UTF8

```
1    INSERT INTO `words`
2                (`word`)
3    VALUES      ('Груша'),
4                ('Яблоко'),
5                ('Апельсин')
```

Уже на этом этапе мы получаем сообщение об ошибке:

MySQL — Сообщение об ошибке

```
1    Error Code: 1366. Incorrect string value:
2    '\xD0\x93\xD1\x80\xD1\x83...' for column 'word' at row 1
```

При этом можно смело утверждать, что нам ещё очень повезло. При иных стечениях обстоятельств могла возникнуть ситуация, в которой вставка данных проходит успешно, а вот упорядочивание и поиск «не работают», т.е. выдают неправильные результаты.

А всего-то нужно было не полениться и указать в своём инструменте проектирования кодировку символов, чтобы запрос на создание таблицы выглядел так (обратите внимание на строки 5 и 6 запроса):

MySQL — Правильное создание таблицы (кодировки указаны)

```
1    CREATE TABLE `words`
2    (
3     `word` VARCHAR(255) NULL
4    )
5    DEFAULT CHARACTER SET = utf8
6    COLLATE = utf8_general_ci
```

[207] Пожалуйста, никогда так не делайте. Если вы занимаетесь разработкой программного обеспечения и/или баз данных, всё программное обеспечение на вашем компьютере должно быть на английском языке. Без исключений.

Теперь запросы на добавление, упорядочивание, поиск и т.д. будут выдавать правильные ожидаемые результаты.

Да, эту проблему очень легко заметить. Но если вы создаёте приложение, многими копиями которого будет пользоваться большое количество ваших клиентов (и вы не можете заранее знать настройки их СУБД в каждом конкретном случае), готовьтесь к шквалу гневных сообщений о том, что «ничего не работает». Или настраивайте кодировки везде, где это только возможно.

Отдельно обращаюсь к тем, у кого «всегда всё работало и без этого». Нет. Оно не работало, оно «срабатывало». Т.е. до сих пор вам просто везло. Теперь вы знаете, как повысить надёжность разрабатываемых вами решений, а как будете делать именно вы — решать, конечно же, вам.

Методы доступа

Методы доступа уже были упомянуты ранее{32}, и здесь мы лишь отметим, что к ним в полной мере относятся все проблемы и рекомендации, только что описанные на примере кодировок.

Если полагаться на настройки по умолчанию, легко оказаться в ситуации, когда поведение СУБД будет совсем не таким, какое вы ожидали. Могут возникнуть проблемы с производительностью, контролем ссылочной целостности (да, до сих пор существуют методы доступа, не поддерживающие такой контроль[208]), транзакциями, блокировками таблиц при операциях модификации, репликациями, масштабированием и т.д. и т.п.

Метод доступа всецело определяет поведение базы данных на самом низком уровне — на уровне файлов, в которых хранятся данные. Потому крайне неосмотрительно было бы полагаться на некие умолчания. Напротив — всегда стоит указывать метод доступа явно, если выбранная вами СУБД это позволяет.

Например, в MySQL за это отвечает параметр **`ENGINE`** в синтаксисе создания таблицы. Вы лишь должны явно указать его с использованием выбранного вами инструмента проектирования так, чтобы эта информация попала в итоговый SQL-запрос, с помощью которого будет создаваться ваша база данных (см. строку 5 в приведённом ниже примере запроса):

MySQL — Правильное создание таблицы (указан метод доступа)

```
1  CREATE TABLE `words`
2  (
3    `word` VARCHAR(255) NULL
4  )
5  ENGINE = InnoDB
6  DEFAULT CHARACTER SET = utf8
7  COLLATE = utf8_general_ci
```

Индексы

Индексам уже был посвящён обширный раздел данной книги{107}. Из представленной там информации следует, что существует широчайший выбор вариантов как разновидностей индексов, так и их параметров.

Многие индексы (например, уникальные{110} или на полях, по которым очевидно очень часто будет выполняться поиск), как правило, видны ещё на даталогическом уровне моделирования — и там же создаются. Но для того и существует физический уровень, чтобы не только ещё раз проверить, не забыт ли какой-то из «очевидных» индексов, но и выполнить серию дополнительных действий:

- наполнить базу данных тестовыми данными, максимально приближенными к реальным (как по объёму, так и по содержанию);

[208] Например, метод доступа MyISAM в MySQL.

- на основе уже имеющейся информации о типичных запросах провести нагрузочное тестирование{398};
- определить те узкие места{398} в производительности базы данных, которые можно устранить с использованием индексов;
- создать необходимые индексы;
- сделать пометку на будущее о необходимости периодической проверки эффективности созданных индексов как в процессе разработки базы данных и работающих с ней приложений, так и в процессе реальной работы созданного проекта.

Создание и удаление индексов (за исключением уникальных{110} и первичных{111}) влияет не на структуру и логику работы базы данных, а на её производительность. Поэтому вносить соответствующие правки можно смело и в любой момент времени (хоть и предполагается, что при правильном проектировании этот момент наступит раньше, чем с базой данных начнут работать реальные пользователи).

Основная же сложность состоит в том, что при тонкой настройке индексов необходимо не только проводить много экспериментов, но и тщательно изучать техническую документацию к конкретной версии конкретной СУБД, т.к. очень часто знакомые вам по одной ситуации решения ведут себя совершенно иначе в другой ситуации.

Настройки СУБД

Здесь мы достигли предела в невозможности сформулировать хоть какие-нибудь общие рекомендации. Вопрос «как настроить СУБД?» без указания сотен дополнительных деталей звучит так же абстрактно и нелепо, как и, например, вопрос «как приготовить еду?» или «как нарисовать картину?».

Самый честный совет, который можно дать начинающим разработчикам баз данных, звучит так: не меняйте никакие настройки СУБД, если вы не понимаете совершенно чётко, что именно, почему, зачем и как вы делаете. В большинстве случаев настройки СУБД по умолчанию «оптимизированы» под широкий спектр типичных решений, к которым, скорее всего, и относится ваша база данных.

Если же создаваемый вами продукт выходит за рамки «типичных», то вот эти его отличительные особенности вы и должны учитывать, изучая документацию и экспериментируя с настройками СУБД.

И обязательно помните о двух вещах:

- создавайте резервные копии перед каждым внесением изменений;
- многократно всё проверьте на тестовом окружении перед тем, как вносить правки в настройки СУБД, с которой уже работают реальные пользователи.

Задание 4.3.a: сформулируйте список вопросов, ответы на которые помогли бы вам улучшить физический уровень моделирования базы данных «Банк»{413}.

4.3.2. Инструменты и техники проектирования на физическом уровне

В предыдущей главе было отмечено, что на данном уровне моделирования нас интересуют права доступа, кодировки, методы доступа, индексы, настройки СУБД.

Что касается техник и подходов к принятию необходимых решений, то ничего принципиально нового здесь не добавляется — мы всё также должны собирать информацию от:

- заказчика — чтобы максимально полно понять специфику проекта, границы бюджета (это может ощутимо повлиять на выбор доступных нам конфигураций СУБД), типичные сценарии работы с базой данных и т.д.;
- разработчиков взаимодействующего с базой данных приложения (или приложений, если их несколько) — чтобы более точно спрогнозировать типичную форму нагрузки на базу данных, увидеть специфику выполняемых запросов, оценить требования к безопасности и т.д.;
- технических специалистов (если это — не мы) дата-центра (или облачного сервиса), в котором будет расположена СУБД с нашей базой данных — чтобы заблаговременно знать все ключевые ограничения и доступные нам возможности по настройке СУБД и инфраструктуры.

Поскольку интересующая нас информация крайне разнообразна по составу и форме представления, мы можем использовать для её организации и интеграции в модель базы данных любой из рассмотренных ранее в данном разделе инструментов.

Как минимум, управление кодировками, методами доступа и индексами доступно практически в любом инструменте моделирования на даталогическом уровне{305}. Причём те инструменты, которые на даталогическом уровне были отмечены как излишне узкоспециализированные{306}, здесь, напротив, могут оказаться наиболее полезными.

Что касается управления правами доступа и настройками СУБД, то самым выгодным инструментом здесь будет такой, который позволит вам автоматически настраивать эти параметры каждый раз при генерации базы данных и/или развёртывании её в СУБД. Т.е. такой инструмент должен уметь автоматически выполнять в нужный момент времени набор SQL-запросов, модифицировать текстовые файлы, вносить изменения в реестр Windows и т.д.

Всеми этими (и множеством других) возможностями обладают DevOps[209]-инструменты. Поскольку их количество огромно, а скорость их развития и изменения колоссальна, список конкретных рекомендованных инструментов будет устаревать буквально каждую неделю, но ничто не мешает вам в любой момент времени обратиться за рекомендациями к более опытным коллегам или элементарно «загуглить» наиболее оптимальный для вашей конкретной ситуации инструмент.

Несколько наглядных примеров будет представлено в следующей главе, а в завершение этой представим небольшую «шпаргалку», которая обобщает материал двух последних глав в краткой форме и также содержит общие советы (которые в наибольшей степени пригодятся начинающим).

[209] **DevOps** — a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality. («Wikipedia») [https://en.wikipedia.org/wiki/DevOps]

Что	Когда начинать обдумывать	Когда заканчивать реализовывать	Инструменты	Общий совет
Права доступа	На инфологическом уровне	К моменту ввода БД в эксплуатацию	SQL-скрипты + DevOps-инструменты	Если вы не создаёте по-настоящему сложный и масштабный проект, реализуйте представленный на рисунке 4.3.a{315} подход и сэкономьте силы и время
Кодировки	На инфологическом уровне	К моменту приёмочного тестирования	Средство проектирования, используемое на даталогическом уровне	Обязательно прописывайте кодировки явным образом везде, где это позволяет выбранная вами СУБД
Методы доступа	На даталогическом уровне	К моменту приёмочного тестирования	Средство проектирования, используемое на даталогическом уровне	Обязательно прописывайте методы доступа явным образом везде, где это позволяет выбранная вами СУБД
Индексы	На даталогическом уровне	Можно изменять даже в процессе эксплуатации СУБД	Средство проектирования, используемое на даталогическом уровне + SQL-скрипты + DevOps-инструменты + средства нагрузочного тестирования	Обязательно проводите исследования (в т.ч. нагрузочное тестирование), не полагайтесь на то, что некоторые решения будут просто казаться логичными — здесь может быть много сюрпризов
Настройки СУБД	На даталогическом уровне	Можно изменять даже в процессе эксплуатации СУБД	SQL-скрипты + DevOps-инструменты	Не меняйте никакие настройки СУБД, если вы не понимаете совершенно чётко, что именно, почему, зачем и как вы делаете

Итак, переходим к примерам.

Задание 4.3.b: доработайте с использованием Sparx Enterprise Architect модель базы данных «Банк»{409} так, чтобы отразить в ней все необходимые нюансы физического уровня проектирования (см. раздаточный материал{4}).

4.3.3. Пример проектирования на физическом уровне

На физическом уровне довольно сложно визуализировать как процесс проектирования, так и его результат. Тем более, что наша учебная база данных часто не будет нуждаться в каких-то особых доработках. Но мы постараемся раскрыть весь процесс настолько полно, насколько это можно сделать в виде текста и картинок.

Права доступа

Да, в нашем конкретном случае намного рациональнее использовать работу с СУБД от имени одного пользователя (см. рисунок 4.3.a{315}), но если бы мы решили пойти по более сложному пути (см. рисунок 4.3.b{316}), реализовать это можно было бы следующим образом.

В первую очередь необходимо определить некие глобальные роли пользователей и перечень их прав относительно объектов базы данных.

Для краткости будем использовать общепринятые сокращения, произошедшие от известной аббревиатуры CRUD[210]: C — право на создание записи, R — право на чтение записи, U — право на обновление записи, D — право на удаление записи.

	Приложение	Гость	Пользователь	Модератор	Администратор
age_restriction	R	R	R	CRUD	CRUD
ban	R	R	R	R	CRUD
bonus	R	R	R	R	CRUD
comment	R	R	R	CRUD	CRUD
download_link	R	R	R	CRUD	CRUD
file	R	R	CRUD	CRUD	CRUD
file_category	R	R	R	CRUD	CRUD
file_permission	R	R	CRUD	CRUD	CRUD
group	R	R	R	CRUD	CRUD
ip_blacklist	CRUD	-	-	-	CRUD
log	C	-	-	-	CRUD
log_archive	C	-	-	-	CRUD
m2m_file_file_category	R	R	CRUD	CRUD	CRUD
m2m_user_group	R	R	R	CRUD	CRUD
m2m_user_role	R	R	R	CRUD	CRUD
mark	R	R	CRUD	CRUD	CRUD
operation	R	-	-	-	CRUD
page	R	R	R	CRUD	CRUD
permission	R	R	R	R	CRUD
role	R	R	R	R	CRUD
statistics	R	R	R	R	CRUD
user	CRUD	R	CRUD	CRUD	CRUD

Очевидно, что часть операций (например, протоколирование) приложение должно выполнять в любом случае — вне зависимости от того, какой пользователь с ним сейчас работает — потому нам придётся создать отдельную роль (и отдельного пользователя) именно для самого приложения. От имени этого же пользователя будет происходить регистрация остальных пользователей.

Теперь необходимо создать соответствующий SQL-код. Для краткости приведём таковой для ролей «Приложение» и «Администратор» (создание аналогичного кода для ролей «Гость», «Пользователь» и «Модератор» оставим на задание для самостоятельной проработки 4.3.d{331}).

[210] CRUD — Create, Read, Update, Delete (создать, прочитать, обновить, удалить).