Введение в системы управления базами данных
А.Ю.Пушников

Данное учебное пособие было опубликовано в 1999 г. в двух частях издательством Башкирского государственного университета. Выходные данные: УДК 519.6, ББК 32.973-018.2 Пушников А.Ю. Введение в системы управления базами данных. Часть 1. Реляционная модель данных: Учебное пособие/Изд-е Башкирского ун-та. - Уфа, 1999. - 108 с. - ISBN 5-7477-0350-1. В электронной версии - полный текст книги

Web-страница: http://www.citforum.ru/database/dblearn/index.shtml

НОРМАЛЬНЫЕ ФОРМЫ ОТНОШЕНИЙ

Этапы разработки базы данных

Целью разработки любой базы данных является хранение и использование информации о какой-либо предметной области. Для реализации этой цели имеются следующие инструменты:

1. Реляционная модель данных - удобный способ представления данных предметной области.
2. Язык SQL - универсальный способ манипулирования такими данными.

Однако очевидно, что для одной и той же предметной области реляционные отношения можно спроектировать множеством различных способов. Например, можно спроектировать несколько отношений с большим количеством атрибутов, или наоборот, разнести все атрибуты по большому числу мелких отношений. Как определить, по каким признакам нужно помещать атрибуты в те или иные отношения?

В данной главе рассматриваются способы "хорошего" или "правильного" проектирования реляционных отношений. Сначала мы обсудим, что значит "хорошие" или "правильные" модели данных. Потом будут введены понятия первой, второй и третьей нормальных форм отношений (1НФ, 2НФ, 3НФ) и показано, что "хорошими" являются отношения в третьей нормальной форме.

При разработке базы данных обычно выделяется несколько уровней моделирования, при помощи которых происходит переход от предметной области к конкретной реализации базы данных средствами конкретной СУБД. Можно выделить следующие уровни:

- Сама предметная область
- Модель предметной области
- Логическая модель данных
- Физическая модель данных
- Собственно база данных и приложения

Предметная область - это часть реального мира, данные о которой мы хотим отразить в базе данных. Например, в качестве предметной области можно выбрать бухгалтерию какого-либо предприятия, отдел кадров, банк, магазин и т.д. Предметная область бесконечна и содержит как

существенно важные понятия и данные, так и малозначащие или вообще не значащие данные. Так, если в качестве предметной области выбрать учет товаров на складе, то понятия "накладная" и "счет-фактура" являются существенно важными понятиями, а то, что сотрудница, принимающая накладные, имеет двоих детей - это для учета товаров неважно. Однако, с точки зрения отдела кадров данные о наличии детей являются существенно важными. Таким образом, важность данных зависит от выбора предметной области.

Модель предметной области. Модель предметной области - это наши знания о предметной области. Знания могут быть как в виде неформальных знаний в мозгу эксперта, так и выражены формально при помощи каких-либо средств. В качестве таких средств могут выступать текстовые описания предметной области, наборы должностных инструкций, правила ведения дел в компании и т.п. Опыт показывает, что текстовый способ представления модели предметной области крайне неэффективен. Гораздо более информативными и полезными при разработке баз данных являются описания предметной области, выполненные при помощи специализированных графических нотаций. Имеется большое количество методик описания предметной области. Из наиболее известных можно назвать методику структурного анализа SADT и основанную на нем IDEF0, диаграммы потоков данных Гейна-Сарсона, методику объектно-ориентированного анализа UML, и др. Модель предметной области описывает скорее процессы, происходящие в предметной области и данные, используемые этими процессами. От того, насколько правильно смоделирована предметная область, зависит успех дальнейшей разработки приложений.

Логическая модель данных. На следующем, более низком уровне находится логическая модель данных предметной области. Логическая модель описывает понятия предметной области, их взаимосвязь, а также ограничения на данные, налагаемые предметной областью. Примеры понятий - "сотрудник", "отдел", "проект", "зарплата". Примеры взаимосвязей между понятиями - "сотрудник числится ровно в одном отделе", "сотрудник может выполнять несколько проектов", "над одним проектом может работать несколько сотрудников". Примеры ограничений - "возраст сотрудника не менее 16 и не более 60 лет".

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но *без привязки к конкретной СУБД*. Более того, логическая модель данных необязательно должна быть выражена средствами именно *реляционной* модели данных. Основным средством разработки логической модели данных в настоящий момент являются различные варианты ***ER-диаграмм*** (***Entity-Relationship***, ***диаграммы сущность-связь***). Одну и ту же ER-модель можно преобразовать как в реляционную модель данных, так и в модель данных для иерархических и сетевых СУБД, или в постреляционную модель данных. Однако, т.к. мы рассматриваем именно реляционные СУБД, то можно считать, что логическая модель данных для нас формулируется в терминах реляционной модели данных.

Решения, принятые на предыдущем уровне, при разработке модели предметной области, определяют некоторые границы, в пределах которых можно развивать логическую модель данных, в пределах же этих границ можно принимать различные решения. Например, модель предметной области складского учета содержит понятия "склад", "накладная", "товар". При разработке соответствующей реляционной модели эти термины обязательно должны быть использованы, но различных способов реализации тут много - можно создать одно отношение, в котором будут присутствовать в качестве атрибутов "склад", "накладная", "товар", а можно создать три отдельных отношения, по одному на каждое понятие.

При разработке логической модели данных возникают вопросы: хорошо ли спроектированы отношения? Правильно ли они отражают модель предметной области, а следовательно, и саму предметную область?

Физическая модель данных. На еще более низком уровне находится физическая модель данных. Физическая модель данных описывает данные средствами конкретной СУБД. Мы будем считать, что физическая модель данных реализована средствами именно *реляционной* СУБД, хотя, как уже сказано выше, это необязательно. Отношения, разработанные на стадии формирования логической модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц, для ключевых атрибутов создаются уникальные индексы, домены преображаются в типы данных, принятые в конкретной СУБД.

Ограничения, имеющиеся в логической модели данных, реализуются различными средствами СУБД, например, при помощи индексов, декларативных ограничений целостности, триггеров, хранимых процедур. При этом опять-таки решения, принятые на уровне логического моделирования, определяют некоторые границы, в пределах которых можно развивать физическую модель данных. Точно также, в пределах этих границ можно принимать различные решения. Например, отношения, содержащиеся в логической модели данных, должны быть преобразованы в таблицы, но для каждой таблицы можно дополнительно объявить различные индексы, повышающие скорость обращения к данным. Многое тут зависит от конкретной СУБД.

При разработке физической модели данных возникают вопросы: хорошо ли спроектированы таблицы? Правильно ли выбраны индексы? Насколько много программного кода в виде триггеров и хранимых процедур необходимо разработать для поддержания целостности данных?

Собственно база данных и приложения. И, наконец, как результат предыдущих этапов появляется, собственно, сама база данных. База данных реализована на конкретной программно-аппаратной основе, и выбор этой основы позволяет существенно повысить скорость работы с базой данных. Например, можно выбирать различные типы компьютеров, менять количество процессоров, объем оперативной памяти, дисковые подсистемы и т.п. Очень большое значение имеет также настройка СУБД в пределах выбранной программно-аппаратной платформы.

Но опять решения, принятые на предыдущем уровне - уровне физического проектирования, определяют границы, в пределах которых можно принимать решения по выбору программно-аппаратной платформы и настройки СУБД.

Таким образом ясно, что решения, принятые на каждом этапе моделирования и разработки базы данных, будут сказываться на дальнейших этапах. Поэтому особую роль играет принятие правильных решений *на ранних этапах моделирования*.

Критерии оценки качества логической модели данных

Цель данной главы - описать некоторые принципы построения *хороших логических моделей данных*. Хороших в том смысле, что решения, принятые в процессе логического проектирования, приводили бы к хорошим физическим моделям и в конечном итоге к хорошей работе базы данных.

Для того чтобы оценить качество принимаемых решений на уровне логической модели данных, необходимо сформулировать некоторые критерии качества в терминах физической модели и конкретной реализации и посмотреть, как различные решения, принятые в процессе *логического* моделирования, влияют на качество *физической* модели и на скорость работы базы данных.

Конечно, таких критериев может быть очень много и выбор их в достаточной степени произволен. Мы рассмотрим некоторые из таких критериев, которые являются безусловно важными с точки зрения получения качественной базы данных:

- Адекватность базы данных предметной области
- Легкость разработки и сопровождения базы данных
- Скорость выполнения операций обновления данных (вставка, обновление, удаление кортежей)
- Скорость выполнения операций выборки данных

Адекватность базы данных предметной области

База данных должна адекватно отражать предметную область. Это означает, что должны выполняться следующие условия:

1. Состояние базы данных в каждый момент времени должно соответствовать состоянию предметной области.
2. Изменение состояния предметной области должно приводить к соответствующему изменению состояния базы данных
3. Ограничения предметной области, отраженные в модели предметной области, должны, некоторым образом, отражаться и учитываться базе данных.

Легкость разработки и сопровождения базы данных

Практически любая база данных, за исключением совершенно элементарных, содержит некоторое количество программного кода в виде триггеров и хранимых процедур.

Хранимые процедуры - это процедуры и функции, хранящиеся непосредственно в базе данных в откомпилированном виде и которые могут запускаться пользователями или приложениями, работающими с базой данных. Хранимые процедуры обычно пишутся либо на специальном процедурном расширении языка SQL (например, PL/SQL для ORACLE или Transact-SQL для MS SQL Server), или на некотором универсальном языке программирования, например, C++, с включением в код операторов SQL в соответствии со специальными правилами такого включения. Основное назначение хранимых процедур - реализация бизнес-процессов предметной области.

Триггеры - это хранимые процедуры, связанные с некоторыми событиями, происходящими во время работы базы данных. В качестве таких событий выступают операции вставки, обновления и удаления строк таблиц. Если в базе данных определен некоторый триггер, то он запускается *автоматически* всегда при возникновении события, с которым этот триггер связан. Очень важным является то, что пользователь не может обойти триггер. Триггер срабатывает независимо от того, кто из пользователей и каким способом инициировал событие, вызвавшее запуск триггера. Таким образом, основное назначение триггеров - автоматическая поддержка

целостности базы данных. Триггеры могут быть как достаточно простыми, например, поддерживающими ссылочную целостность, так и довольно сложными, реализующими какие-либо сложные ограничения предметной области или сложные действия, которые должны произойти при наступлении некоторых событий. Например, с операцией вставки нового товара в накладную может быть связан триггер, который выполняет следующие действия - проверяет, есть ли необходимое количество товара, при наличии товара добавляет его в накладную и уменьшает данные о наличии товара на складе, при отсутствии товара формирует заказ на поставку недостающего товара и тут же посылает заказ по электронной почте поставщику.

Очевидно, что чем больше программного кода в виде триггеров и хранимых процедур содержит база данных, тем сложнее ее разработка и дальнейшее сопровождение.

Скорость операций обновления данных (вставка, обновление, удаление)

На уровне логического моделирования мы определяем реляционные отношения и атрибуты этих отношений. На этом уровне мы не можем определять какие-либо физические структуры хранения (индексы, хеширование и т.п.). Единственное, чем мы можем управлять - это распределением атрибутов по различным отношениям. Можно описать мало отношений с большим количеством атрибутов, или много отношений, каждое из которых содержит мало атрибутов. Таким образом, необходимо попытаться ответить на вопрос - влияет ли количество отношений и количество атрибутов в отношениях на скорость выполнения операций обновления данных. Такой вопрос, конечно, не является достаточно корректным, т.к. скорость выполнения операций с базой данных сильно зависит от физической реализации базы данных. Тем не менее, попытаемся *качественно* оценить это влияние при *одинаковых подходах к физическому моделированию*.

Основными операциями, изменяющими состояние базы данных, являются операции вставки, обновления и удаления записей. В базах данных, требующих постоянных изменений (складской учет, системы продаж билетов и т.п.) производительность определяется скоростью выполнения большого количества небольших операций вставки, обновления и удаления.

Рассмотрим операцию вставки записи в таблицу. Вставка записи производится в одну из свободных страниц памяти, выделенной для данной таблицы. СУБД постоянно хранит информацию о наличии и расположении свободных страниц. Если для таблицы не созданы индексы, то операция вставки выполняется фактически с одинаковой скоростью независимо от размера таблицы и от количества атрибутов в таблице. Если в таблице имеются индексы, то при выполнении операции вставки записи индексы должны быть перестроены. Таким образом, скорость выполнения операции вставки *уменьшается при увеличении количества индексов* у таблицы и *мало зависит от числа строк* в таблице.

Рассмотрим операции обновления и удаления записей из таблицы. Прежде, чем обновить или удалить запись, ее необходимо найти. Если таблица не индексирована, то единственным способом поиска является последовательное сканирование таблицы в поиске нужной записи. В этом случае, скорость операций обновления и удаления существенно увеличивается с увеличением количества записей в таблице и не зависит от количества атрибутов. Но на самом деле неиндексированные таблицы практически никогда не используются. Для каждой таблицы обычно объявляется один или несколько индексов, соответствующий потенциальным ключам.

При помощи этих индексов поиск записи производится очень быстро и практически не зависит от количества строк и атрибутов в таблице (хотя, конечно, некоторая зависимость имеется). Если для таблицы объявлено несколько индексов, то при выполнении операций обновления и удаления эти индексы должны быть перестроены, на что тратится дополнительное время. Таким образом, скорость выполнения операций обновления и удаления также *уменьшается при увеличении количества индексов* у таблицы и *мало зависит от числа строк* в таблице.

Можно предположить, что чем больше атрибутов имеет таблица, тем больше для нее будет объявлено индексов. Эта зависимость, конечно, не прямая, но *при одинаковых подходах* к физическому моделированию обычно так и происходит. Таким образом, можно принять допущение, что *чем больше атрибутов имеют отношения,* разработанные в ходе логического моделирования, *тем медленнее будут выполняться операции обновления данных*, за счет затраты времени на перестройку большего количества индексов.

Дополнительные соображения в пользу приведенного тезиса о замедлении выполнения операций обновления данных (влияние журнализации, длины строк таблиц) приведены в работе А.Прохорова [27].

Скорость операций выборки данных

Одно из назначений базы данных - предоставление информации пользователям. Информация извлекается из реляционной базы данных при помощи оператора SQL - SELECT. Одной из наиболее дорогостоящих операций при выполнении оператора SELECT является операция соединение таблиц. Таким образом, чем больше взаимосвязанных отношений было создано в ходе логического моделирования, тем больше вероятность того, что при выполнении запросов эти отношения будут соединяться, и, следовательно, тем медленнее будут выполняться запросы. Таким образом, увеличение количества отношений приводит к замедлению выполнения операций выборки данных, особенно если запросы заранее неизвестны.

Основной пример

Рассмотрим в качестве предметной области некоторую организацию, выполняющую некоторые проекты. Модель предметной области опишем следующим неформальным текстом:

1. Сотрудники организации выполняют проекты.
2. Проекты состоят из нескольких заданий.
3. Каждый сотрудник может участвовать в одном или нескольких проектах, или временно не участвовать ни в каких проектах.
4. Над каждым проектом может работать несколько сотрудников, или временно проект может быть приостановлен, тогда над ним не работает ни один сотрудник.
5. Над каждым заданием в проекте работает ровно один сотрудник.
6. Каждый сотрудник числится в одном отделе.
7. Каждый сотрудник имеет телефон, находящийся в отделе сотрудника.

В ходе дополнительного уточнения того, какие данные необходимо учитывать, выяснилось следующее:

1. О каждом сотруднике необходимо хранить табельный номер и фамилию. Табельный номер является уникальным для каждого сотрудника.
2. Каждый отдел имеет уникальный номер.
3. Каждый проект имеет номер и наименование. Номер проекта является уникальным.
4. Каждая работа из проекта имеет номер, уникальный в пределах проекта. Работы в разных проектах могут иметь одинаковые номера.

1НФ (Первая Нормальная Форма)

Понятие первой нормальной формы уже обсуждалось в главе 2. ***Первая нормальная форма*** (***1НФ***) - это обычное отношение. Согласно нашему определению отношений, любое отношение автоматически уже находится в 1НФ. Напомним кратко свойства отношений (это и будут свойства 1НФ):

- В отношении нет одинаковых кортежей.
- Кортежи не упорядочены.
- Атрибуты не упорядочены и различаются по наименованию.
- Все значения атрибутов атомарны.

В ходе логического моделирования на первом шаге предложено хранить данные в одном отношении, имеющем следующие атрибуты:

СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ (*Н_СОТР*, ФАМ, Н_ОТД, ТЕЛ, *Н_ПРО*, ПРОЕКТ, Н_ЗАДАН)

где

Н_СОТР - табельный номер сотрудника

ФАМ - фамилия сотрудника

Н_ОТД - номер отдела, в котором числится сотрудник

ТЕЛ - телефон сотрудника

Н_ПРО - номер проекта, над которым работает сотрудник

ПРОЕКТ - наименование проекта, над которым работает сотрудник

Н_ЗАДАН - номер задания, над которым работает сотрудник

Т.к. каждый сотрудник в каждом проекте выполняет ровно одно задание, то в качестве потенциального ключа отношения необходимо взять пару атрибутов {***Н_СОТР***, ***Н_ПРО***}.

В текущий момент состояние предметной области отражается следующими фактами:

- Сотрудник Иванов, работающий в 1 отделе, выполняет в первом проекте "Космос" задание 1 и во втором проекте "Климат" задание 1.

- Сотрудник Петров, работающий в 1 отделе, выполняет в первом проекте "Космос" задание 2.
- Сотрудник Сидоров, работающий во 2 отделе, выполняет в первом проекте "Космос" задание 3 и во втором проекте "Климат" задание 2.

Это состояние отражается в таблице (курсивом выделены ключевые атрибуты):

Н_СОТР	**ФАМ**	**Н_ОТД**	**ТЕЛ**	***Н_ПРО***	**ПРОЕКТ**	**Н_ЗАДАН**
1	Иванов	1	11-22-33	*1*	Космос	1
1	Иванов	1	11-22-33	*2*	Климат	1
2	Петров	1	11-22-33	*1*	Космос	2
3	Сидоров	2	33-22-11	*1*	Космос	3
3	Сидоров	2	33-22-11	*2*	Климат	2

Таблица 1 Отношение СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ

Аномалии обновления

Даже одного взгляда на таблицу отношения **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** достаточно, чтобы увидеть, что данные хранятся в ней с *большой избыточностью*. Во многих строках повторяются фамилии сотрудников, номера телефонов, наименования проектов. Кроме того, в данном отношении хранятся вместе независимые друг от друга данные - и данные о сотрудниках, и об отделах, и о проектах, и о работах по проектам. Пока никаких действий с отношением не производится, это не страшно. Но как только состояние предметной области изменяется, то, при попытках соответствующим образом изменить состояние базы данных, возникает большое количество проблем.

Исторически эти проблемы получили название ***аномалии обновления***. Попытки дать строгое понятие аномалии в базе данных не являются вполне удовлетворительными [51, 7]. В данных работах аномалии определены как противоречие между моделью предметной области и физической моделью данных, поддерживаемых средствами конкретной СУБД. "Аномалии возникают в том случае, когда наши знания о предметной области оказываются, по каким-то причинам, невыразимыми в схеме БД или входящими в противоречие с ней" [7]. Мы придерживаемся другой точки зрения, заключающейся в том, что аномалий в смысле определений упомянутых авторов нет, а есть либо *неадекватность модели данных* предметной области, либо некоторые *дополнительные трудности в реализации ограничений* предметной области средствами СУБД. Более глубокое обсуждение проблемы строгого определения понятия аномалий выходит за пределы данной работы.

Таким образом, мы будем придерживаться интуитивного понятия аномалии как неадекватности модели данных предметной области, (что говорит на самом деле о том, что логическая модель данных попросту неверна!) или как необходимости дополнительных усилий для реализации всех ограничений определенных в предметной области (дополнительный программный код в виде триггеров или хранимых процедур).

Т.к. аномалии проявляют себя при выполнении операций, изменяющих состояние базы данных, то различают следующие виды аномалий:

- Аномалии вставки (INSERT)
- Аномалии обновления (UPDATE)
- Аномалии удаления (DELETE)

В отношении **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** можно привести примеры следующих аномалий:

Аномалии вставки (INSERT)

В отношение **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** нельзя вставить данные о сотруднике, который пока не участвует ни в одном проекте. Действительно, если, например, во втором отделе появляется новый сотрудник, скажем, Пушников, и он пока не участвует ни в одном проекте, то мы должны вставить в отношение кортеж (4, Пушников, 2, 33-22-11, null, null, null). Это сделать невозможно, т.к. атрибут ***Н_ПРО*** (номер проекта) входит в состав потенциального ключа, и, следовательно, не может содержать null-значений.

Точно также нельзя вставить данные о проекте, над которым пока не работает ни один сотрудник.

Причина аномалии - хранение в одном отношении разнородной информации (и о сотрудниках, и о проектах, и о работах по проекту).

Вывод - логическая модель данных неадекватна модели предметной области. База данных, основанная на такой модели, будет работать неправильно.

Аномалии обновления (UPDATE)

Фамилии сотрудников, наименования проектов, номера телефонов повторяются во многих кортежах отношения. Поэтому если сотрудник меняет фамилию, или проект меняет наименование, или меняется номер телефона, то такие изменения необходимо *одновременно* выполнить во всех местах, где эта фамилия, наименование или номер телефона встречаются, иначе отношение станет некорректным (например, один и тот же проект в разных кортежах будет называться по-разному). Таким образом, обновление базы данных одним действием реализовать невозможно. Для поддержания отношения в целостном состоянии необходимо написать триггер, который при обновлении одной записи корректно исправлял бы данные и в других местах.

Причина аномалии - избыточность данных, также порожденная тем, что в одном отношении хранится разнородная информация.

Вывод - увеличивается сложность разработки базы данных. База данных, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

Аномалии удаления (DELETE)

При удалении некоторых данных может произойти потеря другой информации. Например, если закрыть проект "Космос" и удалить все строки, в которых он встречается, то будут потеряны все данные о сотруднике Петрове. Если удалить сотрудника Сидорова, то будет потеряна информация о том, что в отделе номер 2 находится телефон 33-22-11. Если по проекту временно прекращены работы, то при удалении данных о работах по этому проекту будут удалены и данные о самом проекте (наименование проекта). При этом если был сотрудник, который работал только над этим проектом, то будут потеряны и данные об этом сотруднике.

Причина аномалии - хранение в одном отношении разнородной информации (и о сотрудниках, и о проектах, и о работах по проекту).

Вывод - логическая модель данных неадекватна модели предметной области. База данных, основанная на такой модели, будет работать неправильно.

Функциональные зависимости

Отношение **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** находится в 1НФ, при этом, как было показано выше, логическая модель данных не адекватна модели предметной области. Таким образом, первой нормальной формы *недостаточно* для правильного моделирования данных.

Определение функциональной зависимости

Для устранения указанных аномалий (а на самом деле *для правильного проектирования модели данных*!) применяется метод нормализации отношений. Нормализация основана на понятии функциональной зависимости атрибутов отношения.

Определение 1. Пусть R - отношение. Множество атрибутов Y ***функционально зависимо*** от множества атрибутов X (X ***функционально определяет*** Y) тогда и только тогда, когда *для любого состояния* отношения R для любых кортежей $r_1, r_2 \in R$ из того, что $r_1.X = r_2.X$ следует что $r_1.Y = r_2.Y$ (т.е. во всех кортежах, имеющих одинаковые значения атрибутов X, значения атрибутов Y также совпадают *в любом состоянии* отношения R). Символически функциональная зависимость записывается

$X \to Y$.

Множество атрибутов X называется ***детерминантом функциональной зависимости***, а множество атрибутов Y называется ***зависимой частью***.

Замечание. Если атрибуты X составляют потенциальный ключ отношения R, то *любой* атрибут отношения R функционально зависит от X.

Пример 1. В отношении **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** можно привести следующие примеры функциональных зависимостей:

Зависимость атрибутов от ключа отношения:

{*Н_СОТР, Н_ПРО*} $\rightarrow$**ФАМ**

{*Н_СОТР, Н_ПРО*} $\rightarrow$**Н_ОТД**

{*Н_СОТР, Н_ПРО*} $\rightarrow$**ТЕЛ**

{*Н_СОТР, Н_ПРО*} $\rightarrow$**ПРОЕКТ**

{*Н_СОТР, Н_ПРО*} $\rightarrow$**Н_ЗАДАН**

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:

Н_СОТР $\rightarrow$**ФАМ**

Н_СОТР $\rightarrow$**Н_ОТД**

Н_СОТР $\rightarrow$**ТЕЛ**

Зависимость наименования проекта от номера проекта:

Н_ПРО $\rightarrow$**ПРОЕКТ**

Зависимость номера телефона от номера отдела:

Н_ОТД $\rightarrow$**ТЕЛ**

Замечание. Приведенные функциональные зависимости *не выведены* из внешнего вида отношения, приведенного в таблице 1. Эти зависимости отражают взаимосвязи, обнаруженные между объектами предметной области, и являются дополнительными ограничениями, определяемыми предметной областью. Таким образом, функциональная зависимость - *семантическое понятие*. Она возникает, когда по значениям одних данных в предметной области можно определить значения других данных. Например, зная табельный номер сотрудника, можно определить его фамилию, по номеру отдела можно определить телефона. Функциональная зависимость *задает дополнительные ограничения* на данные, которые могут храниться в отношениях. Для корректности базы данных (адекватности предметной области) необходимо при выполнении операций модификации базы данных проверять все ограничения, определенные функциональными зависимостями.

Функциональные зависимости отношений и математическое понятие функциональной зависимости

Функциональная зависимость атрибутов отношения напоминает понятие функциональной зависимости в математике. *Но это не одно и то же*. Для сравнения напомним математическое понятие функциональной зависимости:

Определение 2. Функциональная зависимость (***функция***) - это тройка объектов (X,Y,f), где

X - множество (***область определения***),

Y - множество (***множество значений***),

f - правило, согласно которому каждому элементу $x \in X$ ставится в соответствие один и только один элемент $y \in Y$ (***правило функциональной зависимости***).

Функциональная зависимость обычно обозначается как $f : X \to Y$ или $y = f(x)$.

Замечание. Правило f может быть задано любым способом - в виде формулы (чаще всего), при помощи таблицы значений, при помощи графика, текстовым описанием и т.д.

Функциональная зависимость атрибутов отношения тоже *напоминает* это определение. Действительно:

- В качестве области определения выступает домен, на котором определен атрибут X (или декартово произведение доменов, если X является множеством атрибутов)
- В качестве множества значений выступает домен, на котором определен атрибут Y (или декартово произведение доменов)
- Правило f реализуется следующим алгоритмом - 1) по данному значению атрибута X найти *любой* кортеж отношения, содержащий это значение, 2) значение атрибута Y в этом кортеже и будет значением функциональной зависимости, соответствующим данному X. Определение функциональной зависимости в отношении гарантирует, что найденное значение Y *не зависит от выбора кортежа*, поэтому правило f определено корректно.

Отличие от математического понятия отношения состоит в том, что, если рассматривать математическое понятие функции, то *для фиксированного значения* $x \in X$ соответствующее значение функции $y = f(x)$ *всегда одно и то же*. Например, если задана функция $y = x^2$, то для значения $x = 2$ соответствующее значение y *всегда* будет равно 4. В противоположность этому в отношениях значение зависимого атрибута может принимать *различные* значения в различных состояниях базы данных. Например, атрибут **ФАМ** функционально зависит от атрибута **Н_СОТР**. Предположим, что сейчас сотрудник с табельным номером 1 имеет фамилию Иванов, т.е. при значении детерминанта равного 1, значение зависимого аргумента равно "Иванов". Но сотрудник может сменить фамилию, например на "Сидоров". Теперь *при том же* значении детерминанта, равного 1, значение зависимого аргумента равно "Сидоров".

Таким образом, понятие функциональной зависимости атрибутов нельзя считать полностью эквивалентным математическому понятию функциональной зависимости, т.к. значение этой зависимости различны при разных состояниях отношения, и, самое главное, эти значения могут меняться *непредсказуемо*.

Функциональная зависимость атрибутов утверждает лишь то, что для каждого конкретного состояния базы данных по значению одного атрибута (детерминанта) можно однозначно

определить значение другого атрибута (зависимой части). Но конкретные значение зависимой части могут быть различны в различных состояниях базы данных.

2НФ (Вторая Нормальная Форма)

Определение 3. Отношение R находится во ***второй нормальной форме*** (***2НФ***) тогда и только тогда, когда отношение находится в 1НФ и *нет неключевых атрибутов, зависящих от части сложного ключа*. (***Неключевой атрибут*** - это атрибут, не входящий в состав никакого потенциального ключа).

Замечание. Если потенциальный ключ отношения является простым, то отношение автоматически находится в 2НФ.

Отношение **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** не находится в 2НФ, т.к. есть атрибуты, зависящие от части сложного ключа:

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника является зависимостью от части сложного ключа:

Н_СОТР $\rightarrow$**ФАМ**

Н_СОТР $\rightarrow$**Н_ОТД**

Н_СОТР $\rightarrow$**ТЕЛ**

Зависимость наименования проекта от номера проекта является зависимостью от части сложного ключа:

Н_ПРО $\rightarrow$**ПРОЕКТ**

Для того, чтобы устранить зависимость атрибутов от части сложного ключа, нужно произвести ***декомпозицию*** отношения на несколько отношений. При этом *те атрибуты, которые зависят от части сложного ключа,* выносятся в отдельное отношение.

Отношение **СОТРУДНИКИ_ОТДЕЛЫ_ПРОЕКТЫ** декомпозируем на три отношения - **СОТРУДНИКИ_ОТДЕЛЫ**, **ПРОЕКТЫ**, **ЗАДАНИЯ**.

Отношение **СОТРУДНИКИ_ОТДЕЛЫ** (***Н_СОТР***, **ФАМ**, **Н_ОТД**, **ТЕЛ**):

Функциональные зависимости:

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:

Н_СОТР $\rightarrow$**ФАМ**

Н_СОТР $\rightarrow$**Н_ОТД**

Н_СОТР $\rightarrow$**ТЕЛ**

Зависимость номера телефона от номера отдела:

Н_ОТД →ТЕЛ

Н_СОТР	**ФАМ**	**Н_ОТД**	**ТЕЛ**
1	Иванов	1	11-22-33
2	Петров	1	11-22-33
3	Сидоров	2	33-22-11

Таблица 2 Отношение СОТРУДНИКИ_ОТДЕЛЫ

Отношение **ПРОЕКТЫ** (***Н_ПРО***, **ПРОЕКТ**):

Функциональные зависимости:

Н_ПРО **→ПРОЕКТ**

Н_ПРО	**ПРОЕКТ**
1	Космос
2	Климат

Таблица 3 Отношение ПРОЕКТЫ

Отношение **ЗАДАНИЯ** (***Н_СОТР***, ***Н_ПРО***, **Н_ЗАДАН**):

Функциональные зависимости:

{*Н_СОТР, Н_ПРО*} →Н_ЗАДАН

Н_СОТР	***Н_ПРО***	**Н_ЗАДАН**
1	*1*	1
1	*2*	1
2	*1*	2
3	*1*	3
3	*2*	2

Таблица 4 Отношения ЗАДАНИЯ

Анализ декомпозированных отношений

Отношения, полученные в результате декомпозиции, находятся в 2НФ. Действительно, отношения **СОТРУДНИКИ_ОТДЕЛЫ** и **ПРОЕКТЫ** имеют простые ключи, следовательно

автоматически находятся в 2НФ, отношение **ЗАДАНИЯ** имеет сложный ключ, но единственный неключевой атрибут **Н_ЗАДАН** функционально зависит от всего ключа **{*Н_СОТР, Н_ПРО*}**.

Часть аномалий обновления устранена. Так, данные о сотрудниках и проектах теперь хранятся в различных отношениях, поэтому при появлении сотрудников, не участвующих ни в одном проекте просто добавляются кортежи в отношение **СОТРУДНИКИ_ОТДЕЛЫ**. Точно также, при появлении проекта, над которым не работает ни один сотрудник, просто вставляется кортеж в отношение **ПРОЕКТЫ**.

Фамилии сотрудников и наименования проектов теперь хранятся без избыточности. Если сотрудник сменит фамилию или проект сменит наименование, то такое обновление будет произведено в одном месте.

Если по проекту временно прекращены работы, но требуется, чтобы сам проект сохранился, то для этого проекта удаляются соответствующие кортежи в отношении **ЗАДАНИЯ**, а данные о самом проекте и данные о сотрудниках, участвовавших в проекте, остаются в отношениях **ПРОЕКТЫ** и **СОТРУДНИКИ_ОТДЕЛЫ**.

Тем не менее, часть аномалий разрешить не удалось.

Оставшиеся аномалии вставки (INSERT)

В отношение **СОТРУДНИКИ_ОТДЕЛЫ** нельзя вставить кортеж (4, Пушников, 1, 33-22-11), т.к. при этом получится, что два сотрудника из 1-го отдела (Иванов и Пушников) имеют разные номера телефонов, а это противоречит модели предметной области. В этой ситуации можно предложить два решения, в зависимости от того, что реально произошло в предметной области. Другой номер телефона может быть введен по двум причинам - по ошибке человека, вводящего данные о новом сотруднике, или потому что номер в отделе действительно изменился. Тогда можно написать триггер, который при вставке записи о сотруднике проверяет, совпадает ли телефон с уже имеющимся телефоном у другого сотрудника этого же отдела. Если номера отличаются, то система должна задать вопрос, оставить ли старый номер в отделе или заменить его новым. Если нужно оставить старый номер (новый номер введен ошибочно), то кортеж с данными о новом сотруднике будет вставлен, но номер телефона будет у него будет тот, который уже есть в отделе (в данном случае, 11-22-33). Если же номер в отделе действительно изменился, то кортеж будет вставлен с новым номером, и одновременно будут изменены номера телефонов у всех сотрудников этого же отдела. И в том и в другом случае не обойтись без разработки громоздкого триггера.

Причина аномалии - избыточность данных, порожденная тем, что в одном отношении хранится разнородная информация (о сотрудниках и об отделах).

Вывод - увеличивается сложность разработки базы данных. База данных, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

Оставшиеся аномалии обновления (UPDATE)

Одни и те же номера телефонов повторяются во многих кортежах отношения. Поэтому если в отделе меняется номер телефона, то такие изменения необходимо одновременно выполнить во всех местах, где этот номер телефона встречаются, иначе отношение станет некорректным. Таким образом, обновление базы данных одним действием реализовать невозможно. Необходимо написать триггер, который при обновлении одной записи корректно исправляет номера телефонов в других местах.

Причина аномалии - избыточность данных, также порожденная тем, что в одном отношении хранится разнородная информация.

Вывод - увеличивается сложность разработки базы данных. База данных, основанная на такой модели, будет работать правильно только при наличии дополнительного программного кода в виде триггеров.

Оставшиеся аномалии удаления (DELETE)

При удалении некоторых данных по-прежнему может произойти потеря другой информации. Например, если удалить сотрудника Сидорова, то будет потеряна информация о том, что в отделе номер 2 находится телефон 33-22-11.

Причина аномалии - хранение в одном отношении разнородной информации (и о сотрудниках, и об отделах).

Вывод - логическая модель данных неадекватна модели предметной области. База данных, основанная на такой модели, будет работать неправильно.

Заметим, что при переходе ко второй нормальной форме отношения стали *почти* адекватными предметной области. Остались также трудности в разработке базы данных, связанные с необходимостью написания триггеров, поддерживающих целостность базы данных. Эти трудности теперь связаны только с одним отношением **СОТРУДНИКИ_ОТДЕЛЫ**.

3НФ (Третья Нормальная Форма)

Определение 4. Атрибуты называются ***взаимно независимыми***, если ни один из них не является функционально зависимым от другого.

Определение 5. Отношение R находится в ***третьей нормальной форме*** (***3НФ***) тогда и только тогда, когда отношение находится в 2НФ и *все неключевые атрибуты взаимно независимы*.

Отношение **СОТРУДНИКИ_ОТДЕЛЫ** не находится в 3НФ, т.к. имеется функциональная зависимость неключевых атрибутов (зависимость номера телефона от номера отдела):

Н_ОТД $\rightarrow$ТЕЛ

Для того, чтобы устранить зависимость неключевых атрибутов, нужно произвести декомпозицию отношения на несколько отношений. При этом *те неключевые атрибуты, которые являются зависимыми,* выносятся в отдельное отношение.

Отношение **СОТРУДНИКИ_ОТДЕЛЫ** декомпозируем на два отношения - **СОТРУДНИКИ**, **ОТДЕЛЫ**.

Отношение **СОТРУДНИКИ** (***Н_СОТР***, **ФАМ**, **Н_ОТД**):

Функциональные зависимости:

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:

Н_СОТР →**ФАМ**

Н_СОТР →**Н_ОТД**

Н_СОТР →**ТЕЛ**

Н_СОТР	**ФАМ**	**Н_ОТД**
1	Иванов	1
2	Петров	1
3	Сидоров	2

Таблица 5 Отношение СОТРУДНИКИ

Отношение **ОТДЕЛЫ** (***Н_ОТД***, **ТЕЛ**):

Функциональные зависимости:

Зависимость номера телефона от номера отдела:

Н_ОТД →**ТЕЛ**

Н_ОТД	**ТЕЛ**
1	11-22-33
2	33-22-11

Таблица 6 Отношение ОТДЕЛЫ

Обратим внимание на то, что атрибут **Н_ОТД**, *не являвшийся ключевым* в отношении **СОТРУДНИКИ_ОТДЕЛЫ**, *становится потенциальным ключом* в отношении **ОТДЕЛЫ**. Именно за счет этого устраняется избыточность, связанная с многократным хранением одних и тех же номеров телефонов.

Вывод. Таким образом, все обнаруженные аномалии обновления устранены. Реляционная модель, состоящая из четырех отношений **СОТРУДНИКИ**, **ОТДЕЛЫ**, **ПРОЕКТЫ**, **ЗАДАНИЯ**, находящихся в третьей нормальной форме, является адекватной описанной модели предметной области, и требует наличия только тех триггеров, которые поддерживают ссылочную целостность. Такие триггеры являются стандартными и не требуют больших усилий в разработке.

Алгоритм нормализации (приведение к 3НФ)

Итак, алгоритм нормализации (т.е. алгоритм приведения отношений к 3НФ) описывается следующим образом.

Шаг 1 (Приведение к 1НФ). На первом шаге задается одно или несколько отношений, отображающих понятия предметной области. По модели предметной области (не по внешнему виду полученных отношений!) выписываются обнаруженные функциональные зависимости. Все отношения автоматически находятся в 1НФ.

Шаг 2 (Приведение к 2НФ). Если в некоторых отношениях обнаружена зависимость атрибутов от части сложного ключа, то проводим декомпозицию этих отношений на несколько отношений следующим образом: те атрибуты, которые зависят от части сложного ключа выносятся в отдельное отношение вместе с этой частью ключа. В исходном отношении остаются все ключевые атрибуты:

Исходное отношение: $R(K_1, K_2, A_1, .., A_n, B_1, \ldots, B_m)$.

Ключ: (K_1, K_2) - сложный.

Функциональные зависимости:

$(K_1, K_2) \to (A_1, .., A_n, B_1, \ldots B_m)$ - зависимость всех атрибутов от ключа отношения.

$(K_1) \to (A_1, .., A_n)$ - зависимость некоторых атрибутов от части сложного ключа.

Декомпозированные отношения:

$R_1(K_1, K_2, B_1, \ldots, B_m)$ - остаток от исходного отношения. Ключ (K_1, K_2).

$R_2(K_1, A_1, .., A_n)$ - атрибуты, вынесенные из исходного отношения вместе с частью сложного ключа. Ключ K_1.

Шаг 3 (Приведение к 3НФ). Если в некоторых отношениях обнаружена зависимость некоторых неключевых атрибутов других неключевых атрибутов, то проводим декомпозицию этих отношений следующим образом: те неключевые атрибуты, которые зависят других неключевых атрибутов выносятся в отдельное отношение. В новом отношении ключом становится детерминант функциональной зависимости:

Исходное отношение: $R(K,A_1,..,A_n,B_1,...,B_m)$.

Ключ: K.

Функциональные зависимости:

$K \to \{A_1,..,A_n,B_1,...,B_m\}$ - зависимость всех атрибутов от ключа отношения.

$\{A_1,..,A_n\} \to \{B_1,...,B_m\}$ - зависимость некоторых неключевых атрибутов других неключевых атрибутов.

Декомпозированные отношения:

$R_1(K,A_1,..,A_n)$ - остаток от исходного отношения. Ключ K.

$R_2(A_1,..,A_n,B_1,...,B_m)$ - атрибуты, вынесенные из исходного отношения вместе с детерминантом функциональной зависимости. Ключ $\{A_1,..,A_n\}$.

Замечание. На практике, при создании логической модели данных, как правило, не следуют прямо приведенному алгоритму нормализации. Опытные разработчики обычно сразу строят отношения в 3НФ. Кроме того, основным средством разработки логических моделей данных являются различные варианты ER-диаграмм. Особенность этих диаграмм в том, что они сразу позволяют создавать отношения в 3НФ. Тем не менее, приведенный алгоритм важен по двум причинам. *Во-первых*, этот алгоритм показывает, какие проблемы возникают при разработке слабо нормализованных отношений. *Во-вторых*, как правило, модель предметной области никогда не бывает правильно разработана с первого шага. Эксперты предметной области могут забыть о чем-либо упомянуть, разработчик может неправильно понять эксперта, во время разработки могут измениться правила, принятые в предметной области, и т.д. Все это может привести к появлению новых зависимостей, которые отсутствовали в первоначальной модели предметной области. Тут как раз и необходимо использовать алгоритм нормализации хотя бы для того, чтобы убедиться, что отношения остались в 3НФ и логическая модель не ухудшилась.

Анализ критериев для нормализованных и ненормализованных моделей данных

Сравнение нормализованных и ненормализованных моделей

Соберем воедино результаты анализа критериев, по которым мы хотели оценить влияние логического моделирования данных на качество физических моделей данных и производительность базы данных:

Критерий	Отношения слабо нормализованы (1НФ, 2НФ)	Отношения сильно нормализованы (3НФ)
Адекватность базы данных предметной области	ХУЖЕ (-)	ЛУЧШЕ (+)
Легкость разработки и сопровождения базы данных	СЛОЖНЕЕ (-)	ЛЕГЧЕ (+)
Скорость выполнения вставки, обновления, удаления	МЕДЛЕННЕЕ (-)	БЫСТРЕЕ (+)
Скорость выполнения выборки данных	БЫСТРЕЕ (+)	МЕДЛЕННЕЕ (-)

Как видно из таблицы, более сильно нормализованные отношения оказываются лучше спроектированы (три плюса, один минус). Они больше соответствуют предметной области, легче в разработке, для них быстрее выполняются операции модификации базы данных. Правда, это достигается ценой некоторого замедления выполнения операций выборки данных.

У слабо нормализованных отношений единственное преимущество - если к базе данных обращаться только с запросами на выборку данных, то для слабо нормализованных отношений такие запросы выполняются быстрее. Это связано с тем, что в таких отношениях уже как бы произведено соединение отношений и на это не тратится время при выборке данных.

Таким образом, выбор степени нормализации отношений зависит от характера запросов, с которыми чаще всего обращаются к базе данных.

OLTP и OLAP-системы

Можно выделить некоторые классы систем, для которых больше подходят сильно или слабо нормализованные модели данных.

Сильно нормализованные модели данных хорошо подходят для так называемых ***OLTP-приложений*** (***On-Line Transaction Processing*** (***OLTP***)- ***оперативная обработка транзакций***). Типичными примерами OLTP-приложений являются системы складского учета, системы заказов билетов, банковские системы, выполняющие операции по переводу денег, и т.п. Основная функция подобных систем заключается в выполнении большого количества коротких транзакций. Сами транзакции выглядят относительно просто, например, "снять сумму денег со счета А, добавить эту сумму на счет В". Проблема заключается в том, что, во-первых, транзакций очень много, во-вторых, выполняются они одновременно (к системе может быть подключено несколько тысяч одновременно работающих пользователей), в-третьих, при возникновении ошибки, транзакция должна целиком откатиться и вернуть систему к состоянию, которое было до начала транзакции (не должно быть ситуации, когда деньги сняты со счета А, но не поступили на счет В). Практически все запросы к базе данных в OLTP-приложениях состоят из команд вставки, обновления, удаления. Запросы на выборку в основном предназначены для предоставления пользователям возможности выбора из различных

справочников. Большая часть запросов, таким образом, известна заранее еще на этапе проектирования системы. Таким образом, критическим для OLTP-приложений является скорость и надежность выполнения коротких операций обновления данных. Чем выше уровень нормализации данных в OLTP-приложении, тем оно, как правило, быстрее и надежнее. Отступления от этого правила могут происходить тогда, когда уже на этапе разработки известны некоторые часто возникающие запросы, требующие соединения отношений и от скорости выполнения которых существенно зависит работа приложений. В этом случае можно пожертвовать нормализацией для ускорения выполнения подобных запросов.

Другим типом приложений являются так называемые ***OLAP-приложения*** (***On-Line Analitical Processing*** (***OLAP***) - ***оперативная аналитическая обработка данных***). Это обобщенный термин, характеризующий принципы построения ***систем поддержки принятия решений*** (***Decision Support System - DSS***), ***хранилищ данных*** (***Data Warehouse***), ***систем интеллектуального анализа данных*** (***Data Mining***). Такие системы предназначены для нахождения зависимостей между данными (например, можно попытаться определить, как связан объем продаж товаров с характеристиками потенциальных покупателей), для проведения анализа "что если…". OLAP-приложения оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми их электронных таблиц или из других источников данных. Такие системы характеризуются следующими признаками:

- Добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из OLTP-приложения).
- Данные, добавленные в систему, обычно никогда не удаляются.
- Перед загрузкой данные проходят различные процедуры "очистки", связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления для одних и тех же понятий, данные могут быть некорректны, ошибочны.
- Запросы к системе являются нерегламентированными и, как правило, достаточно сложными. Очень часто новый запрос формулируется аналитиком для уточнения результата, полученного в результате предыдущего запроса.
- Скорость выполнения запросов важна, но не критична.

Данные OLAP-приложений обычно представлены в виде одного или нескольких гиперкубов, измерения которого представляют собой справочные данные, а в ячейках самого гиперкуба хранятся собственно данные. Например, можно построить гиперкуб, измерениями которого являются: время (в кварталах, годах), тип товара и отделения компании, а в ячейках хранятся объемы продаж. Такой гиперкуб будет содержать данных о продажах различных типов товаров по кварталам и подразделениям. Основываясь на этих данных, можно отвечать на вопросы вроде "у какого подразделения самые лучшие объемы продаж в текущем году?", или "каковы тенденции продаж отделений Юго-Западного региона в текущем году по сравнению с предыдущим годом?"

Физически гиперкуб может быть построен на основе специальной ***многомерной модели данных*** (***MOLAP - Multidimensional OLAP***) или построен средствами реляционной модели данных (***ROLAP - Relational OLAP***).

Возвращаясь к проблеме нормализации данных, можно сказать, что в системах OLAP, использующих реляционную модель данных (ROLAP), данные целесообразно хранить в виде

слабо нормализованных отношений, содержащих заранее вычисленные основные итоговые данные. Большая избыточность и связанные с ней проблемы тут не страшны, т.к. обновление происходит только в момент загрузки новой порции данных. При этом происходит как добавление новых данных, так и пересчет итогов.

Корректность процедуры нормализации - декомпозиция без потерь. Теорема Хеза

Как было показано выше, алгоритм нормализации состоит в выявлении функциональных зависимостей предметной области и соответствующей декомпозиции отношений. Предположим, что мы уже имеем работающую систему, в которой накоплены данные. Пусть данных корректны в текущий момент, т.е. факты предметной области правильно отражаются текущим состоянием базы данных. Если в предметной области обнаружена новая функциональная зависимость (либо она была пропущена на этапе моделирования предметной области, либо просто изменилась предметная область), то возникает необходимость заново нормализовать данные. При этом некоторые отношения придется декомпозировать в соответствии с алгоритмом нормализации. Возникают естественные вопросы - что произойдет с уже накопленными данными? Не будут ли данные потеряны в ходе декомпозиции? Можно ли вернуться обратно к исходным отношениям, если будет принято решение отказаться от декомпозиции, восстановятся ли при этом данные?

Для ответов на эти вопросы нужно ответить на вопрос - что же представляет собой декомпозиция отношений с точки зрения операций реляционной алгебры? При декомпозиции мы из одного отношения получаем два или более отношений, каждое из которых содержит часть атрибутов исходного отношения. В полученных новых отношениях необходимо удалить дубликаты строк, если таковые возникли. Это в точности означает, что декомпозиция отношения есть не что иное, как взятие одной или нескольких проекций исходного отношения так, чтобы эти проекции в совокупности содержали (возможно, с повторениями) *все* атрибуты исходного отношения. Т.е., при декомпозиции *не должны теряться атрибуты* отношений. Но при декомпозиции также не должны потеряться и сами данные. Данные можно считать не потерянными в том случае, если возможна обратная операция - по декомпозированным отношениям можно восстановить исходное отношение *в точности в прежнем виде*. Операцией, обратной операции проекции, является операция соединения отношений. Имеется большое количество видов операции соединения (см. гл. 4). Т.к. при восстановлении исходного отношения путем соединения проекций не должны появиться новые атрибуты, то необходимо использовать *естественное соединение*.

Определение 6. Проекция $R[X]$ отношения R на множество атрибутов X называется ***собственной***, если множество атрибутов X является *собственным подмножеством* множества атрибутов отношения R (т.е. множество атрибутов X не совпадает с множеством *всех* атрибутов отношения R).

Определение 7. *Собственные* проекции R_1 и R_2 отношения R называются ***декомпозицией без потерь***, если отношение R *точно восстанавливается* из них при помощи естественного соединения *для любого состояния* отношения R:

$$R_1 \text{ JOIN } R_2 = R.$$

Рассмотрим пример, показывающий, что декомпозиция без потерь происходит не всегда.

Пример 2. Пусть дано отношение R:

НОМЕР	ФАМИЛИЯ	ЗАРПЛАТА
1	Иванов	1000
2	Петров	1000

Таблица 7 Отношение R

Рассмотрим первый вариант декомпозиции отношения R на два отношения:

НОМЕР	ЗАРПЛАТА
1	1000
2	1000

Таблица 8 Отношение R_1

ФАМИЛИЯ	ЗАРПЛАТА
Иванов	1000
Петров	1000

Таблица 9 Отношение R_2

Естественное соединение этих проекций, имеющих общий атрибут "ЗАРПЛАТА", очевидно, будет следующим (каждая строка одной проекции соединится с каждой строкой другой проекции):

НОМЕР	ФАМИЛИЯ	ЗАРПЛАТА
1	Иванов	1000
1	Петров	1000
2	Иванов	1000
2	Петров	1000

Таблица 10 Отношение $R_1 \text{ JOIN } R_2 \neq R$

Итак, данная декомпозиция не является декомпозицией без потерь, т.к. исходное отношение *не восстанавливается в точном виде* по проекциям (серым цветом выделены лишние кортежи).

Рассмотрим другой вариант декомпозиции:

НОМЕР	ФАМИЛИЯ
1	Иванов
2	Петров

Таблица 11 Отношение R_1

НОМЕР	ЗАРПЛАТА
1	1000
2	1000

Таблица 12 Отношение R_2

По данным проекциям, имеющие общий атрибут "НОМЕР", исходное отношение восстанавливается в точном виде. Тем не менее, нельзя сказать, что данная декомпозиция является декомпозицией без потерь, т.к. мы рассмотрели только *одно конкретное состояние* отношения R, и не можем сказать, будет ли и в других состояниях отношение R восстанавливаться точно. Например, предположим, что отношение R перешло в состояние:

НОМЕР	ФАМИЛИЯ	ЗАРПЛАТА
1	Иванов	1000
2	Петров	1000
2	Сидоров	2000

Таблица 13 Отношение R

Кажется, что этого не может быть, т.к. значения в атрибуте "НОМЕР" повторяются. Но мы же ничего не говорили о ключе этого отношения! Сейчас проекции будут иметь вид:

НОМЕР	ФАМИЛИЯ
1	Иванов
2	Петров
2	Сидоров

Таблица 14 Отношение R_1

НОМЕР	ЗАРПЛАТА
1	1000
2	1000
2	2000

Таблица 15 Отношение R_2

Естественное соединение этих проекций будет содержать лишние кортежи:

НОМЕР	ФАМИЛИЯ	ЗАРПЛАТА
1	Иванов	1000
2	Петров	1000
2	Петров	2000
2	Сидоров	1000
2	Сидоров	2000

Таблица 16 Отношение $R_1 \text{ JOIN } R_2 \neq R$

Вывод. Таким образом, *без дополнительных ограничений* на отношение R нельзя говорить о декомпозиции без потерь.

Такими дополнительными ограничениями и являются функциональные зависимости. Имеет место следующая теорема Хеза [54]:

Теорема (Хеза). Пусть $R(A,B,C)$ является отношением, и A,B,C - атрибуты или множества атрибутов этого отношения. Если имеется функциональная зависимость $A \to B$, то проекции $R_1 = R[A,B]$ и $R_2 = R[A,C]$ образуют декомпозицию без потерь.

Доказательство. Необходимо доказать, что $R_1 \text{ JOIN } R_2 = R$ *для любого состояния* отношения R. В левой и правой части равенства стоят *множества* кортежей, поэтому для доказательства достаточно доказать два включения для двух *множеств* кортежей: $R_1 \text{ JOIN } R_2 \supseteq R$ и $R_1 \text{ JOIN } R_2 \subseteq R$.

Докажем первое включение. Возьмем произвольный кортеж $r = (a,b,c) \in R$. Докажем, что он включается также и в $R_1 \text{ JOIN } R_2$. По определению проекции, кортежи $r_1 = (a,b) \in R_1$ и $r_2 = (a,c) \in R_2$. По определению естественного соединения кортежи r_1 и r_2, имеющие одинаковое значение a общего атрибута A, будут соединены в процессе естественного соединения в кортеж $(a,b,c) \in R_1 \text{ JOIN } R_2$. Таким образом, включение доказано.

Докажем обратное включение. Возьмем произвольный кортеж $r = (a,b,c) \in R_1 \text{ JOIN } R_2$. Докажем, что он включается также и в R. По определению естественного соединения получим, что в имеются кортежи $r_1 = (a,b) \in R_1$ и $r_2 = (a,c) \in R_2$. Т.к. $R_1 = R[A,B]$, то существует некоторое значение c_1, такое что кортеж $r_1 = (a,b,c_1) \in R$. Аналогично, существует некоторое значение b_1, такое что кортеж $r_2 = (a,b_1,c) \in R$. Кортежи r_1 и r_2 имеют одинаковое значение атрибута A, равное a. Из этого, в силу функциональной зависимости $A \to B$, следует, что $b = b_1$. Таким образом, кортеж $r_2 = (a,b,c) \in R$. Обратное включение доказано. *Теорема доказана*.

Замечание. В доказательстве теоремы Хеза наличие функциональной зависимости *не использовалось* при доказательстве включения $R_1 \text{ JOIN } R_2 \supseteq R$. Это означает, что при выполнении декомпозиции и последующем восстановлении отношения при помощи естественного соединения, кортежи исходного отношения *не будут потеряны*. Основной смысл теоремы Хеза заключается в доказательстве того, что при этом *не появятся новые кортежи*, отсутствовавшие в исходном отношении.

Т.к. алгоритм нормализации (приведения отношений к 3НФ) основан на имеющихся в отношениях функциональных зависимостях, то теорема Хеза показывает, что алгоритм нормализации является корректным, т.е. в ходе нормализации *не происходит потери информации*.

Выводы

При разработке базы данных можно выделить несколько уровней моделирования:

- Сама предметная область
- Модель предметной области
- Логическая модель данных
- Физическая модель данных
- Собственно база данных и приложения

Ключевые решения, определяющие качество будущей базы данных, закладываются на этапе разработки логической модели данных. "Хорошие" модели данных должны удовлетворять определенным критериям:

- Адекватность базы данных предметной области
- Легкость разработки и сопровождения базы данных
- Скорость выполнения операций обновления данных (вставка, обновление, удаление)
- Скорость выполнения операций выборки данных

Первая нормальная форма (***1НФ***) - это обычное отношение. Отношение в 1НФ обладает следующими свойствами:

- В отношении нет одинаковых кортежей.
- Кортежи не упорядочены.
- Атрибуты не упорядочены.
- Все значения атрибутов атомарны.

Отношения, находящиеся в 1НФ являются "плохими" в том смысле, что они не удовлетворяют выбранным критериям - имеется большое количество аномалий обновления, для поддержания целостности базы данных требуется разработка сложных триггеров.

Отношение R находится во ***второй нормальной форме*** (***2НФ***) тогда и только тогда, когда отношение находится в 1НФ и нет неключевых атрибутов, зависящих от части сложного ключа.

Отношения в 2НФ "лучше", чем в 1НФ, но еще недостаточно "хороши" - остается часть аномалий обновления, по-прежнему требуются триггеры, поддерживающие целостность базы данных.

Отношение R находится в ***третьей нормальной форме*** (***3НФ***) тогда и только тогда, когда отношение находится в 2НФ и все неключевые атрибуты взаимно независимы.

Отношения в 3НФ являются самыми "хорошими" с точки зрения выбранных нами критериев - устранены аномалии обновления, требуются только стандартные триггеры для поддержания ссылочной целостности.

Переход от ненормализованных отношений к отношениям в 3НФ может быть выполнен при помощи ***алгоритма нормализации***. Алгоритм нормализации заключается в последовательной декомпозиции отношений для устранения функциональных зависимостей атрибутов от части сложного ключа (приведение к 2НФ) и устранения функциональных зависимостей неключевых атрибутов друг от друга (приведение к 3НФ).

Корректность процедуры нормализации (декомпозиция без потери информации) доказывается ***теоремой Хеза***.