4 ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

При проектировании базы данных решаются две основные проблемы:

- Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области и было по возможности лучшим (эффективным, удобным и т.д.)? Часто эту проблему называют проблемой логического проектирования баз данных.
- Как обеспечить эффективность выполнения запросов к базе данных, т.е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создание каких дополнительных структур (например, индексов) потребовать и т.д.? Эту проблему называют проблемой физического проектирования баз данных.

В случае реляционных баз данных трудно представить какие-либо общие рецепты по части физического проектирования. Здесь слишком много зависит от используемой СУБД. Более того, мы не будем касаться очень важного аспекта проектирования – определения ограничений целостности (за исключением ограничения первичного ключа). Дело в том, что при использовании СУБД с развитыми механизмами ограничений целостности (например, SQL-ориентированных систем) трудно предложить какой-либо общий подход к определению ограничений целостности. Эти ограничения могут иметь общий вид, и их формулировка пока относится скорее к области искусства, чем инженерного мастерства. Самое большее, что предлагается по этому поводу в литературе, это автоматическая проверка непротиворечивости набора ограничений целостности.

Так что будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том, из каких отношений должна состоять БД и какие атрибуты должны быть у этих отношений.

4.1 Получение реляционной схемы из ER-модели

Шаг 1. Каждая простая сущность превращается в таблицу. Простая сущность – сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

Шаг 2. Каждый атрибут становится возможным столбцом с тем же именем; может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, – не могут.

Шаг 3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификатора, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс

может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

Шаг 4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами. То есть делается копия уникального идентификатора с конца связи «один», и соответствующие столбцы составляют внешний ключ. Необязательные связи соответствуют столбцам, допускающим неопределенные значения; обязательные связи – столбцам, не допускающим неопределенные значения.

Шаг 5. Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Шаг 6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа:

– все подтипы в одной таблице (а);
– для каждого подтипа – отдельная таблица (б).

Все в одной таблице	Таблица – на подтип
Преимущества	
Все хранится вместе Легкий доступ к супертипу и подтипам Требуется меньше таблиц	Более ясны правила подтипов Программы работают только с нужными таблицами
Недостатки	
Слишком общее решение Требуется дополнительная логика работы с разными наборами столбцов и разными ограничениями Потенциальное узкое место (в связи с блокировками) Столбцы подтипов должны быть необязательными В некоторых СУБД для хранения неопределенных значений требуется дополнительная память	Слишком много таблиц Смущающие столбцы в представлении UNION Потенциальная потеря производительности при работе через UNION Над супертипом невозможны модификации

При применении способа (а) таблица создается для наиболее внешнего супертипа, а для подтипов могут создаваться представления. В таблицу добавляется, по крайней мере, один столбец, содержащий код ТИПА; он становится частью первичного ключа.

При использовании метода (б) для каждого подтипа первого уровня (для более нижних – представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы – столбцы супертипа).

Шаг 7. Имеется два способа работы при наличии исключающих связей:

– общий домен (а);
– явные внешние ключи (б).

Если остающиеся внешние ключи все в одном домене, т.е. имеют общий формат (способ (а)), то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. Столбец идентификатора

сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

Общий домен	Явные внешние ключи
Преимущества	
Нужно только два столбца	Условия соединения – явные
Недостатки	
Оба дополнительных атрибута должны использоваться в соединениях	Слишком много столбцов

4.2 Нормализация отношений

Сначала будет рассмотрен классический подход, при котором весь процесс проектирования производится в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений. Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих лучшими свойствами. Процесс проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером набора ограничений является ограничение первой нормальной формы – значения всех атрибутов отношения атомарны. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, мы будем считать, что исходный набор отношений уже соответствует этому требованию.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;

– при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

В основе процесса проектирования лежит метод нормализации: *декомпозиция отношения*, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии *функциональной зависимости*. Для дальнейшего изложения нам потребуются несколько понятий.

Функциональная зависимость. В отношении R атрибут Y функционально зависит от атрибута X (X и Y могут быть составными) в том и только в том случае, если каждому значению X соответствует в точности одно значение Y:

$$R.X \rightarrow R.Y.$$

Полная функциональная зависимость. Функциональная зависимость называется полной, если атрибут Y не зависит функционально от любого подмножества X.

Взаимно независимые атрибуты. Два или более атрибута взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

Возможный ключ. Набор атрибутов X (составной атрибут) называется возможным ключом, если существует полная функциональная зависимость $R.X \rightarrow R.Y$, где Y – набор всех остальных атрибутов отношения R, не вошедших в X.

Таким образом, значения возможного ключа однозначно определяют кортеж отношения R. Полная зависимость означает *минимальность ключа*: при удалении из ключа любого из атрибутов свойство однозначной идентификации кортежа теряется.

Неключевой атрибут. Неключевым атрибутом называется любой атрибут отношения, не входящий в состав ни одного возможного ключа.

Транзитивная функциональная зависимость. Функциональная зависимость $R.X \rightarrow R.Y$ называется транзитивной, если существует такой атрибут Z (возможно, составной), что:

– $Z \not\subset X,\ Y \not\subset Z$;
– существуют зависимости $R.X \rightarrow R.Z$ и $R.Z \rightarrow R.Y$;
– не существует зависимости $R.Z \rightarrow R.X$;

Из последнего требования следует отсутствие зависимости $R.Y \rightarrow R.X$. Это означает отсутствие транзитивной зависимости между возможными ключами.

Первичный ключ. Один из возможных ключей выбирается в качестве основного – *первичного ключа*. Он состоит из минимального количества

атрибутов, как правило – из одного. Атрибуты первичного ключа не могут содержать неопределенных значений.

Рассмотрим следующий пример схемы отношения:

СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ(СОТРУДНИК, ЗАРПЛАТА, ОТДЕЛ, ПРОЕКТ, ЗАДАНИЕ)
Первичный ключ: (СОТРУДНИК, ПРОЕКТ).

Предположим, что существуют следующие функциональные зависимости:

СОТРУДНИК–>ЗАРПЛАТА
СОТРУДНИК–>ОТДЕЛ
ОТДЕЛ–>ЗАРПЛАТА
(СОТРУДНИК, ПРОЕКТ)–>ЗАДАНИЕ

Как видно, хотя первичным ключом является составной атрибут (СОТРУДНИК, ПРОЕКТ), атрибуты ЗАРПЛАТА и ОТДЕЛ функционально зависят от части первичного ключа, атрибута СОТРУДНИК. В результате мы не сможем вставить в отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ кортеж, описывающий сотрудника, который еще не выполняет никакого проекта (первичный ключ не может содержать неопределенное значение). При удалении кортежа мы не только разрушаем связь данного сотрудника с данным проектом, но утрачиваем информацию о том, что он работает в некотором отделе. При переводе сотрудника в другой отдел мы будем вынуждены модифицировать все кортежи, описывающие этого сотрудника, или получим несогласованный результат. Такие неприятные явления называются аномалиями схемы отношения. Они устраняются путем нормализации.

4.1.1 Вторая нормальная форма

Отношение R находится во *второй нормальной форме (2NF)* в том и только в том случае, когда находится в 1NF, и каждый неключевой атрибут полностью зависит от первичного ключа. В этом определении предполагается, что единственным возможным ключом отношения является первичный ключ.

Произведем следующую декомпозицию отношения СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ в два отношения СОТРУДНИКИ-ОТДЕЛЫ и СОТРУДНИКИ-ПРОЕКТЫ:

СОТРУДНИКИ-ОТДЕЛЫ(СОТРУДНИК, ЗАРПЛАТА, ОТДЕЛ)
Первичный ключ: СОТРУДНИК
Функциональные зависимости:
СОТРУДНИК–>ЗАРПЛАТА
СОТРУДНИК–>ОТДЕЛ
ОТДЕЛ–>ЗАРПЛАТА
СОТРУДНИКИ-ПРОЕКТЫ(СОТРУДНИК, ПРОЕКТ, ЗАДАНИЕ)
Первичный ключ: (СОТРУДНИК, ПРОЕКТ).

Функциональные зависимости:
(СОТРУДНИК, ПРОЕКТ)–>СОТР_ЗАДАН

Каждое из этих двух отношений находится в 2NF, и в них устранены отмеченные выше аномалии (легко проверить, что все указанные операции выполняются без проблем).

Если допустить наличие нескольких ключей, то определение 2NF примет следующий вид:

Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда оно находится в 1NF, и каждый неключевой атрибут полностью зависит от каждого ключа R.

Здесь и далее мы не будем приводить примеры для отношений с несколькими ключами. Они слишком громоздки и относятся к ситуациям, редко встречающимся на практике.

4.1.2 Третья нормальная форма

Рассмотрим еще раз отношение СОТРУДНИКИ-ОТДЕЛЫ, находящееся в 2NF. Заметим, что функциональная зависимость СОТРУДНИК–> ЗАРПЛАТА является транзитивной; она является следствием функциональных зависимостей СОТРУДНИК–>ОТДЕЛ и ОТДЕЛ–>ЗАРПЛАТА. Другими словами, заработная плата сотрудника на самом деле является характеристикой не сотрудника, а отдела, в котором он работает. В результате мы не сможем занести в базу данных информацию, характеризующую заработную плату отдела, до тех пор, пока в этом отделе не появится хотя бы один сотрудник (первичный ключ не может содержать неопределенное значение). При удалении кортежа, описывающего последнего сотрудника данного отдела, мы лишимся информации о заработной плате отдела. Чтобы согласованным образом изменить заработную плату отдела, мы будем вынуждены предварительно найти все кортежи, описывающие сотрудников этого отдела. То есть в отношении СОТРУДИКИ-ОТДЕЛЫ по-прежнему существуют аномалии. Их можно устранить путем дальнейшей нормализации.

Отношение R находится в *третьей нормальной форме (3NF)* в том и только в том случае, если находится во 2NF и не содержит транзитивных зависимостей.

Можно произвести декомпозицию отношения СОТРУДНИКИ-ОТДЕЛЫ в два отношения СОТРУДНИКИ и ОТДЕЛЫ:

СОТРУДНИКИ (СОТРУДНИК, ОТДЕЛ)
Первичный ключ: СОТРУДНИК
Функциональные зависимости:
СОТРУДНИК–>ОТДЕЛ
ОТДЕЛЫ(ОТДЕЛ, ЗАРПЛАТА)

Первичный ключ: ОТДЕЛ
Функциональные зависимости:
ОТДЕЛ–>ЗАРПЛАТА

Каждое из этих двух отношений находится в 3NF и свободно от отмеченных аномалий.

Если отказаться от того ограничения, что отношение обладает единственным ключом, то определение 3NF примет следующую форму:

Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится во 2NF, и каждый неключевой атрибут не является транзитивно зависимым от какого-либо ключа R.

На практике третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

4.1.3 Нормальная форма Бойса-Кодда

Рассмотрим следующий пример схемы отношения:

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, СОТР_ФИО, ПРОЕКТ, ЗАДАНИЕ)
Возможные ключи: (СОТР_НОМЕР, ПРОЕКТ), (СОТР_ФИО, ПРОЕКТ)
Функциональные зависимости:
(СОТР_НОМЕР, ПРОЕКТ)–>СОТР_ЗАДАН
(СОТР_ФИО, ПРОЕКТ)–>СОТР_ЗАДАН
СОТР_НОМЕР–>ПРОЕКТ
СОТР_ФИО–>ПРОЕКТ
СОТР_НОМЕР–>СОТР_ФИО
СОТР_ФИО–>СОТР_НОМЕР

В этом примере мы предполагаем, что личность сотрудника полностью определяется как его номером, так и именем (это не вполне корректное предположение, но достаточное для примера).

Отношение СОТРУДНИКИ-ПРОЕКТЫ находится в 3NF. Две последние зависимости означают, что имеются неполные функциональные зависимости от возможного ключа атрибута, который является частью другого возможного ключа. Это также приводит к аномалиям. Например, для того чтобы изменить имя сотрудника с данным номером согласованным образом, нам потребуется модифицировать все кортежи, включающие его номер.

Детерминант. Детерминантом называется любой атрибут, от которого полностью функционально зависит некоторый другой атрибут.

Отношение R находится в *нормальной форме Бойса-Кодда (BCNF)* в том и только в том случае, если оно находится в 3NF и каждый детерминант является возможным ключом.

Очевидно, что это требование не выполнено для отношения СОТРУДНИКИ-ПРОЕКТЫ. Можно произвести его декомпозицию к отношениям СОТРУДНИКИ и СОТРУДНИКИ-ПРОЕКТЫ:

СОТРУДНИКИ(СОТР_НОМЕР, СОТР_ФИО)
Возможные ключи: СОТР_НОМЕР и СОТР_ФИО.
Функциональные зависимости:
СОТР_НОМЕР–>СОТР_ФИО
СОТР_ФИО–>СОТР_НОМЕР
СОТРУДНИКИ-ПРОЕКТЫ(СОТР_НОМЕР, ПРОЕКТ, ЗАДАНИЕ)
Возможный ключ: (СОТР_НОМЕР, ПРОЕКТ)
Функциональные зависимости:
(СОТР_НОМЕР, ПРОЕКТ)–>ЗАДАНИЕ

Возможна альтернативная декомпозиция, если выбрать за основу СОТР_ФИО. В обоих случаях получаемые отношения СОТРУДНИКИ и СОТРУДНИКИ-ПРОЕКТЫ находятся в BCNF, и им не свойственны отмеченные аномалии. Однако первый вариант предпочтительней, т.к. при изменении фамилии сотрудника модифицируется только один кортеж отношения СОТРУДНИКИ.

4.3 Поддержка целостности БД

Целостность (от англ. integrity – нетронутость, неприкосновенность, сохранность, целостность) – понимается как правильность данных в любой момент времени. Но эта цель может быть достигнута лишь в определенных пределах: СУБД не может контролировать правильность каждого отдельного значения, вводимого в базу данных (хотя каждое значение можно проверить на правдоподобность). Например, нельзя обнаружить, что вводимое значение 5 (представляющее номер дня недели) в действительности должно быть равно 3. С другой стороны, значение 9 явно будет ошибочным и СУБД должна его отвергнуть. Однако для этого ей следует сообщить, что номера дней недели должны принадлежать набору (1, 2, 3, 4, 5, 6, 7).

Поддержание целостности базы данных может рассматриваться как защита данных от неверных изменений или разрушений (не путать с незаконными изменениями и разрушениями, являющимися проблемой безопасности). Современные СУБД имеют ряд средств для обеспечения поддержания целостности (так же, как и средств обеспечения поддержания безопасности).

Выделяют три группы правил целостности:

- Целостность по сущностям.
- Целостность по ссылкам.
- Целостность, определяемая пользователем.

В предыдущем подразделе была рассмотрена мотивировка двух правил целостности для обозначений и характеристик, общих для любых реляционных баз данных.

Не допускается, чтобы какой-либо атрибут, участвующий в первичном ключе, принимал неопределенное значение.

Значение внешнего ключа должно либо:

- быть равным значению первичного ключа цели;
- быть полностью неопределенным, т.е. каждое значение атрибута, участвующего во внешнем ключе должно быть неопределенным.

Для любой конкретной базы данных существует ряд дополнительных специфических правил, которые относятся к ней одной и определяются разработчиком. Чаще всего контролируется:

- уникальность тех или иных атрибутов,
- диапазон значений (экзаменационная оценка от 2 до 5),
- принадлежность набору значений (пол «М» или «Ж»).