УДК 681.3

**В.П. Гладков**

Пермский национальный исследовательский
политехнический университет

# УПРАВЛЕНИЕ РИСКАМИ ПРИ ПРОЕКТИРОВАНИИ БАЗ ДАННЫХ

*Рассматривается метод проектирования реляционных баз данных с учетом предыдущего опыта. Несмотря на большое разнообразие предметных областей, количество принимаемых решений конечно. Этот факт может быть использован для ускорения проектирования и повышения его эффективности.*

База данных (БД) как интегрированное хранилище данных разных пользователей, предназначенное для коллективного использования, является неотъемлемой частью любой информационной системы (ИС). Проектирование БД осуществляется на начальной стадии построения ИС, поэтому велика ответственность за неверное решение. К тому же сам процесс проектирования носит неалгоритмический, творческий характер. Для успеха проектирования необходимы богатый жизненный опыт, устойчивые навыки анализа различных предметных областей (ПО).

При анализе ПО возможны следующие риски:

1) пропуск важной объектной сущности;

2) неверная интерпретация связей между сущностями;

3) недостаточная нормализация сущностей, что ведет к аномалиям технологических операций;

4) пропуск важных ограничений целостности;

5) истолкование отдельной сущности как атрибута другой, что приводит к реструктуризации БД в процессе эксплуатации;

6) неверное представление многозначных атрибутов;

7) неверная интерпретация рекурсивных связей.

Для устранения указанных рисков можно выделить в разных ПО ситуации, которые повторяются. Такие ситуации могут быть представлены типовыми решениями.

Первое типовое решение возможно при работе со справочниками. Часто в таблицах оказывается повторяющаяся из некоторого диапазона информация. Ее целесообразно хранить в специальных таблицах-справочниках.

Рассмотрим схему реляционной базы данных (РБД), состоящую из таблиц:

```
Склад(НомерСклада, Название, Город),
Товар(Артикул, Название, Город).
```

Атрибут `Город` повторяется в обеих таблицах и содержит название города. Если средняя длина названия города – 10 символов, количество разных городов – 100, а количество записей в каждой из таблиц – 1000, то для хранения потребуется 1000*2*10 = 20000 байт ~ 19,5 Кб. К тому же при таком способе хранения возникнут аномалии технологических операций, если, например, для каждого города потребуется хранить его код (часть почтового индекса).

Идеальным типовым решением в данном случае является использование третьей справочной таблицы для хранения информации о городах. В этом случае схема РБД преобразуется:

```
Склад(НомерСклада, Название, НомерГорода),
Товар(Артикул, Название, НомерГорода),
СправочникГородов(НомерГорода, Название, Код).
```

Поскольку городов только 100, то номер города может быть двузначным, т.е. занимать один байт, тогда объем справочника без учета кода будет 1*100 + 10*100 = 1100 байт. Использование номера города для связи в таблицах `Склад` и `Товар` потребует еще 2000 байт. Всего для хранения информации о городах в новой схеме БД потребуется: 1100 + 2000 = 3100, что в 6 раз меньше по сравнению с предыдущей схемой.

Использование справочника позволит сэкономить память и устранить аномалии технологических операций.

Подобный прием хранения данных Дж. Дейт называет факторизацией [1].

Часто возникает потребность хранить в БД информацию, которая в ПО имеет структуру таблицы и состоит из «шапки» и тела, которое включает несколько строк (аналог одномерного массива структур). Примерами являются рецепты, накладные, чеки, описи, ведомости инвентаризации и т.п.

Типовым решением в данном случае является использование двух таблиц: первая содержит «шапку», а вторая, связанная с первой связью «один ко многим», – множество строк.

Например, рецепт содержит сведения о больном и враче, а также множество строк с описанием выписанных лекарств. Для отображения рецепта в РБД необходимы следующие таблицы:

Рецепт(НомерРецепта, КомуВыписан, КемВыписан),

СтрокиРецепта(НомерРецепта, НомерСтрокиРецепта, Лекарство, Количество).

В описанной схеме типового решения РБД возникает вопрос о хранении агрегатных данных. Например, следует ли хранить количество наименований выписанных лекарств. С одной стороны, такой атрибут, хранящийся в таблице Рецепт, позволит упростить и ускорить расчеты, с другой стороны, он ведет к дублированию информации, потому что его можно получить на основе таблицы СтрокиРецепта. Для избежания риска дублирования следует отказаться от хранения агрегатного атрибута, а получать его непосредственно перед расчетами по таблице СтрокиРецепта и размещением полученных данных в представлении (View).

Аналогом двумерных массивов является бинарная связь «многие ко многим», для представления ее в РБД используется правило: «Если тип бинарной связи равен *m*:*n*, то для представления данных необходимы три таблицы: по одной для каждого класса, где в качестве ключей используются ключи соответствующих классов, и связующая таблица, в которой помещаются ключи обоих классов. В качестве ключа связующей таблицы выступает комбинация ключей двух первых ключей» [2].

Рассмотрим ПО: «Каждый товар может продаваться в любом магазине, и такие же товары могут продаваться в любом другом магазине».

В ней имеется связь «многие ко многим» между товарами и магазинами. Применение правила приводит к следующим реляционным таблицам:

Товар(Артикул, Название, Город);

Магазин(НомерМагазина, Название, Специализация, Адрес);

ТоварМагазин(Артикул, НомерМагазина, Количество, Цена).

В таблице ТоварМагазин будут повторения номеров магазинов и артикулов, однако комбинации номеров должны быть уникальными. В соответствии с описанием ПО будут существовать товары и магазины, отсутствующие в таблице ТоварМагазин.

Вследствие уникальности комбинации атрибутов Артикул и НомерМагазина операция деления может быть записана на основе сравнения количеств в таблицах «делимое» и «делитель».

Например, запрос «Найдите магазины, в которых продаются все товары» может быть записан так:

```
select НомерМагазина
from ТоварМагазин
group by НомерМагазина
having count(distinct Артикул) = (select count(Артикул) from Товар)
```

Многомерные массивы представляются связями высоких порядков. Общее правило таково: «Если в связь вступают классы объектов *n* типов, то потребуется (*n* + 1) реляционная таблица, *n* таблиц будут хранить информацию о классах объектов, а последняя таблица будет связующей. Ключом связующей таблицы выступает комбинация ключей объектных таблиц».

Формы представления тернарных связей, когда в связь вступают три класса объектов, подробно рассмотрены в [3].

Большым риском является представление симметричных двумерных массивов, например, турнирных таблиц по футболу или таблиц курсов валют.

Пусть необходимо хранить данные об изменении курса валют и периоде их актуальности.

Представление периода актуальности возможно на основе событий или интервального представления.

В представлении на основе событий в базе данных хранятся значение данных, момент времени, когда это значение изменялось, и событие, которое привело к изменению значения.

В интервальном представлении каждое состояние объекта данных снабжается интервалом времени, в течение которого это состояние имело место. Новое состояние помечается интервалом, началом которого является текущая дата, а концом – некоторая дата в далеком будущем, например, 31.12.999.

На основе интервального представления получаем таблицу для хранения курса валют: Exchange_rate(base_code, code, rate, begda, endda) – таблица курсов валют, хранит base_code – код базовой валюты, code – код второй валюты, rate – курс второй валюты по отношению к базовой, begda – дату начала действия курса валюты и endda – дату окончания действия курса валюты.

Последняя таблица устроена так, что хранит избыточную информацию, потому что в ней встречаются, например, такие пары:

```
USD – RUB – курс1,
RUB – USD – 1 / курс1,
USD – EUR – курс2,
EUR – USD – 1 / курс2,
RUB – EUR – курс3,
EUR – RUB – 1 / курс3
```

и им подобные.

Таблица курсов, содержащая такие данные, позволяет просто и быстро отыскать нужную информацию с помощью оператора select, однако возникают аномалии удаления и корректировки, так как удалять и корректировать нужно пару записей.

Хранение таблицы курсов валют, содержащей по одной записи из двух:

```
USD – RUB – курс1,
EUR – USD – 1 / курс2,
RUB – EUR – курс3
```

устраняет аномалии технологических операций, но затрудняет поиск, потому что заранее не известно в каком поле base_code или code необходимо искать данные.

Для устранения описанных трудностей предлагается использовать представление, создаваемое следующим оператором SQL на основе таблицы, описанной последней:

```
CREATE VIEW exchange_rate1 AS
SELECT base_code, code, rate
FROM exchange_rate
UNION
SELECT code, base_code, ( 1 / rate )
FROM exchange_rate
```

Созданное представление `exchange_rate1` хранит по две строки для курсов пары валют и позволяет легко отыскивать нужную информацию, однако базовая таблица хранит по одной строке и, следовательно, легко обновляема.

Придерживаясь рассмотренных типовых решений при проектировании БД, можно существенно уменьшить риски проектирования и получить схему РБД, эффективную и удобную для работы.

**Библиографический список**

1. Дейт Дж. Введение в системы баз данных. – 8-е изд. – М.: Вильямс, 2005.

2. Кренке Д. Теория и практика построения баз данных. – 8-е изд. – СПб.: Питер, 2003.

3. Гладков В.П. К вопросу о классификации тернарных связей // Электротехника, информационные технологии, системы управления: сб. науч. тр. № 4. – Пермь: Изд-во Перм. гос. техн. ун-та, 2010.

Получено 05.09.2011