Базы данных

Юрий Литвинов
yurii.litvinov@gmail.com

12.10.2018г

1. Введение

Без работы с базами данных не обходится ни одно веб-приложение и вообще ни одно приложение, которому надо что-то долго хранить и быстро получать доступ к данным. В принципе, люди долгое время прекрасно обходились обычными файлами, да и для задач типа хранения настроек приложения обычные файлы конфигурации вполне подходят. Но данные часто имеют сложную структуру, к ним приходится выполнять кучу разных видов запросов, к тому же данных, если они заводятся в вашем приложении, быстро становится очень много и они перестают помещаться в оперативке.

Кстати, есть базы данных, а есть системы управления базами данных, нехорошо их путать. БД — это, собственно, данные вместе с их структурой, СУБД — это программа, которая предоставляет к этим данным доступ и позволяет их редактировать (или их структуру). СУБД бывают концептуально разные, каждый подход имеет свои достоинства и недостатки:

- Реляционные — до сих пор самый популярный вид СУБД, использующий реляционную алгебру (https://en.wikipedia.org/wiki/Relational_algebra) для представления данных и операций над ними. Данные в реляционной модели представляются в виде *кортежей* значений разных типов, объединённых в *отношения* — самые настоящие алгебраические отношения и кортежи, в простонародье называемые таблицами и строками. На отношениях определены алгебраические операции, которые принимают отношения и возвращают отношения, определены совершенно формально и их свойства хорошо изучены. Эти алгебраические операции выражаются операторами языка SQL, который достаточно прост, чтобы им мог пользоваться любой школьник, несмотря на то, что внутри происходит кошмарная алгебраическая наука. Операции достаточно выразительны, чтобы с данными можно было делать практически всё, что угодно, и при этом достаточно декларативны, чтобы СУБД могла сама решать, как исполнять запрос, оптимизируя его зачастую в тысячи раз по сравнению с наивной реализацией.

- Объектно-ориентированные — хранят по сути сериализованные объекты и позволяют исполнять запросы в духе “найти объект по шаблону”. Значительно менее выразительный язык запросов компенсируется удобством использования из объектно-ориентированных программ — результатом операции становится не

какой-то невнятный набор кортежей, которые ещё надо как-то загрузить в объектно-ориентированную модель, а самый настоящий объект, будто мы его хранили в памяти. К тому же, такие базы, как правило, проще в конфигурировании и использовании, так что очень популярны сейчас для хранения небольших объёмов данных или данных, не предполагающих сложных запросов.

- Иерархические — СУБД, хранящие данные в виде дерева. Самый типичный нынче пример таких штук — базы, хранящие данные в XML-документах. Для запросов там используется язык XQuery, работающий с выражениями на языке XPath. Для иерархических по своей природе данных такие штуки незаменимы.
- Другие — например, дедуктивные базы данных, которые вообще не хранят данные, они хранят некоторый набор фактов и правила, по которым могут быть выведены остальные факты (например, см. https://en.wikipedia.org/wiki/Datalog).

Наиболее популярны сейчас всё-таки реляционные и объектно-ориентированные базы данных, поэтому часто приходится решать, какую же из этих моделей использовать. Так что рассмотрим соображения, могущие повлиять на решение:

- Реляционные
  - Минусы — сложность интеграции с объектно-ориентированным кодом. Реляционная модель данных концептуально построена по принципу “сущность-связь”, где сущность — это штука, у которой есть имя и атрибуты, могущие иметь значения, а связь — это ссылка из одной сущности на другую. Это очень похоже на объекты и отношения между ними, так что, казалось бы, никаких пробелм быть не должно, но нет: сущности не могут наследоваться, друг от друга, например. Чтобы хранить в реляционной базе объекты двух разных классов, наследующихся от общего предка, потребуется либо две таблицы с атрибутами, куда раскопипащены атрибуты предка, либо три таблицы — для предка и двух потомков, и явные связи между потомками и предком. При загрузке и сохранении таких “объектов” фактически приходится реализовывать наследование вручную. Ещё реляционные базы не могут в отношение “многие ко многим”, потому что связь там — это просто ссылка на строку в другой таблице, она может быть только одной. Для решения этой проблемы заводят вспомогательные “таблицы-развязки”, хранящие в себе только набор связей.
  - Плюсы — выразительный язык запросов, который даже со всякими там таблицами-развязками позволяет выбрать именно те данные, что нужны, и представить их в более-менее удобном виде. Кроме того, запросы выполняются, как правило, очень эффективно, грамотный проектировщик схемы БД может сделать так, чтобы выборка из сотен гигабайт данных занимала всего доли секунды. Впрочем, неудачная схема БД может испортить скорость работы всего приложения очень серьёзно.

2. Реляционная модель данных

Рассмотрим подробнее реляционную модель (без алгебраических деталей). Данные там хранятся в виде отношений, которые удобнее всего себе представлять как таблицы, как на

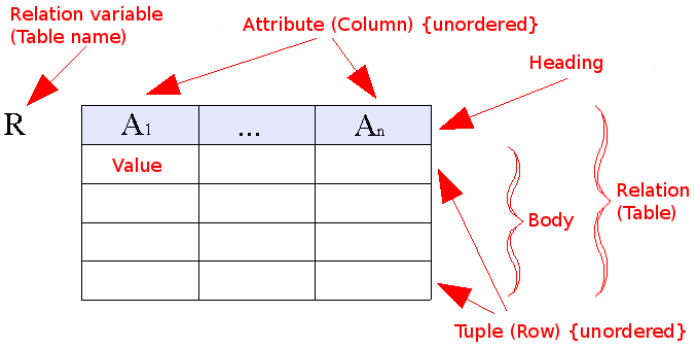

Рис. 1: Отношение.

CustomerID	TaxID	Name	Address
1234567890	555-5512222	Munmun	323 Broadway
2223344556	555-5523232	Wile E.	1200 Main Street
3334445563	555-5533323	Ekta	871 1st Street
423242432	555-5325523	E.F. Codd	123 It Way

Рис. 2: Таблица с данными.

рисунке 1 из Википедии.

У отношения есть имя (имя таблицы), у него есть колонки с именами и типами, и строчки, состоящие из, собственно, данных, лежащих в таблице. Пример таблицы с данными представлен в 2.

Если бы все данные, которые нужны программе, можно было хранить в одной таблице, то никакая реляционная алгебра и не нужна. Интереснее становится, когда данные имеют сложную структуру, требующую нескольких связанных таблиц. Например, список городов и список улиц, где каждая улица привязана к конкретному городу (рис. 3)

Тестовая БД (MariaDB) Устанавливаем MariaDB (https://downloads.mariadb.org/) При установке спросят пароль для пользователя root, придумываем, вводим и запоминаем его Запускаем HeidiSQL, которая поставляется с MariaDB из коробки “Создать” -> “Сеанс в корневой папке” Вводим пароль для рута, который мы запомнили на этапе 1.a. Порт 3306, по умолчанию. Видим список баз, давайте создадим новую Создаём тестовую БД, которую будем мучить Правой кнопкой на Instance сервера (Unnamed), “Создать” -> “База данных” Вводим имя БД, например, “myDB”, жмём “OK” База появилась в списке, создадим таблицу Клик правой кнопкой по базе, “Создать” -> “Таблица” Вводим имя (например, Cities) Жмём “Добавить” рядом со “Столбцы”, вводим имя столбца (id) и тип данных (INT) Снимаем галку “Разрешить NULL” Ставим “По умолчанию” AUTO_INCREMENT Кликаем по нему правой кнопкой, “Создать новый индекс” -> “PRIMARY” Добавляем второй столбец, name (типа VARCHAR(50)) Жмём “Сохранить” Создадим вторую таблицу, People, со столбцами id, name и city_id Не забываем пометить id как AUTO_INCREMENT и PRIMARY Идём во вкладку “Внешние ключи”, жмём “Добавить”, пишем имя ключа

CITY

ID	Name
1	Москва
2	Санкт-Петербург
3	Владивосток

STREET

ID	Name	ID_CITY
181	Малая Бронная	1
182	Тверской Бульвар	1
183	Невский проспект	2
184	Пушкинская	2
185	Светланская	3
186	Пушкинская	3

Рис. 3: Таблицы с городами и улицами.

“city_id”, “Столбцы” – “city_id”, “Справочная таблица” – “cities”, “Внешние столбцы“ – “id” Добавим немного данных “cities” -> “Данные”, правый клик, “Вставить строку”, игнорируем id, вводим “St. Petersburg” в name, кликаем куда-нибудь вне строки, строка добавилась Вставляем ещё Moscow и ещё что-нить “people” -> “Данные”, вставляем так же пару человек Не забываем выбрать city_id из списка, иначе операция добавления не пройдёт и вам напомнят Пробуем написать SQL-запрос Кликаем mydb, вкладку “Запрос” Пишем там “SELECT people.name FROM people, cities WHERE people.city_id = cities.id AND cities.name = ”St. Petersburg”” Жмём “Выполнить SQL” на панели инструментов Видим таблицу с результатами нашего INNER JOIN-запроса снизу.