История

1. Постановка задачи

Проект по созданию СУБД (Системы Управления Базами Данных (DBMS – DataBase Management System)), о котором пойдет речь, начался почти 20 лет назад, в начале 90-х годов прошлого столетия. Сейчас, наверное, это кажется странным, но в то время не существовало SQL DBMS, соответствующих требованиям ACID[1], на платформе Intel-Microsoft (Intel-Novell), а другой вычислительной техники на подавляющем большинстве отечественных предприятиях не было. Создавать же сколько-нибудь серьезные системы, основываясь на «навигационных» «настольных» СУБД типа DBase, FoxPro, Paradox, Clipper, Clarion и т.п., было в принципе невозможно, поскольку они были нетранзакционными, то есть не, обеспечивали одновременную работу нескольких пользователей. Отсутствие многопользовательских СУБД для указанных платформ дополнялось отсутствием литературы, посвященной вопросам их построения, что открывало широкий горизонт для продвижения, развития и апробации собственных идей, недостатка в которых не ощущалось. Данная особенность проекта позволяла довольно просто сформулировать задачу: «Создать многопользовательскую СУБД, отвечающую требованиям стандарта SQL-89[2] и требованиям ACID».

2. Концептуальная основа

В основу проекта был положен системный подход, то есть, мы хорошо осознавали тот факт, что создать надо не программу, а систему. Принципиальное отличие в подходе к разработке программ и систем состоит в том, что система создается по уровням и каждый уровень системы может развиваться независимо от других уровней. Все уровни системы могут разрабатываться одновременно, то есть, параллельно друг другу. Это очень важная особенность, которая позволяет существенно снизить трудоемкость разработки и одновременно повысить эффективность создаваемой системы за счет более глубокой специализации каждого уровня, его проработки от проектирования до реализации[3]. С другой стороны, необходимо очень тщательно продумывать архитектуру системы, до начала проектирования уровней, поскольку последующие изменения в архитектуре могут самым губительным образом сказаться на проекте в целом.

Другой важной особенностью создания систем является то, что система не решает конкретной задачи, в отличие от программы. Система является основой для решения большого круга различных задач, и предоставляет необходимые для решений средства.

Архитектура системы определяет количество уровней, назначение и спецификация характеристик (свойств) для каждого уровня, структуру интерфейсов и протоколы взаимодействия уровней. Интерфейсы группируются по признаку их семантического родства, образуя абстрактные элементы, с которыми взаимодействует над-уровень. Аб-

1 Аббревиатура ACID означает: Atomicity — Атомарность; Consistency — Согласованность; Isolation — Изолированность; Durability — Долговечность.

2 После выхода стандарта SQL-92, проект был переориентирован на соответствие новому стандарту.

3 Более подробно о подходе можно прочитать в материалах: http://www.alexus.ru/russian/articles/design/letter06/index.htm

страктный, в данном контексте, не означает «нереальный» или «идеальный», оно означает, что **любой** реальный элемент, поддерживающий данный набор интерфейсов, может использоваться в данной роли, то есть, использоваться для реальной работы.

На следующем шаге проектирования формируются иерархии объектов на каждом уровне системы. Совершенно необязательно (а иногда и вредно!), чтобы все объекты на всех уровнях обязательно имели какого-то общего для всей системы «предка». В проекте, о котором идет речь, общий «предок» не использовался[4]. Каждый интерфейс порождал свой класс объектов. Разработка каждого класса, как и разработка каждого уровня системы, может происходить независимо и параллельно. Таким образом, весь проект делится на независимые части, что значительно упрощает работу над ним, снижает зависимости одних команд разработчиков от других, повышает ритмичность работы и предсказуемость результатов. Работы над таким проектом легче планировать.

Особое внимание при декомпозиции системы по уровням следует уделить развитию интерфейсов. Каждая группа интерфейсов, при правильной выполненной декомпозиции, имеет свою логику развития. Не следует пытаться реализовать все доступные возможности (все мыслимые интерфейсы). Более правильный подход состоит в том, чтобы определить последовательность разработки тех или иных возможностей, которые могут быть предоставлены данной группой интерфейсов. Другими словами, на предварительной стадии проектирования необходимо проведение семантического анализа, который позволит определить суть/предназначение каждого интерфейса. Уточнение/развитие интерфейсов происходит под влиянием внешних по отношению к системе требований, без искажения сути/предназначения. Совокупность этапов развития интерфейсов определяет этапы развития системы, как единого целого.

Рис. 1. Архитектура системы

Такой подход к разработке (больших) систем является эффективной альтернативой, как методике «waterfall», так и методике «eXtremal Programming» при разработке больших систем.

4 Общий «предок» может использоваться для взаимодействия элементов с системой, то есть, иметь родовые связи с порождающей системой, с целью передачи этих свойств всем последующим классам. На решение практических задач, родовые свойства, как правило, влияния не оказывают.

После окончания проектирования интерфейсов любого из уровней желательно, максимально быстро создать работающий прототип данного интерфейса, то есть, реализовать несколько классов реальных объектов, отвечающих требованиям интерфейса. Это позволит смежным уровням проводить полноценную отладку своей логики. В дальнейшем, классы-прототипы с их простейшей организацией и логикой могут быть заменены более совершенными (например, оптимизированными по тем или иным критериям) классами-версиями-реализациями. Для смежных уровней такая замена происходит незаметно, если не происходит нарушений в спецификациях интерфейсов.

СУБД

Архитектура

Проект был ориентирован на создание реляционной СУБД. В то время начали появляться, так называемые, «постреляционные СУБД» (объектные, объектно-ориентированные, объектно-реляционные и т. п.). Мы были знакомы с некоторыми из них (Versant, ObjectStore, Poet и пр.), но ни конкретные решения, ни их концептуальная основа не выдерживали критики[5]. Было понятно, что это тупиковое направление развития.

«Отец» реляционной модели доктор Е. Кодд представлял базу данных, как множество «отношений» (плоских двумерных таблиц). Отношения могут рассматриваться, как набор «кортежей» (строк таблицы), либо, как набор «атрибутов» (столбцов таблицы). Однотипные атрибуты образуют «домены» - множество допустимых значений атрибутов заданного «базового» типа. Обычно СУБД хранят данные по кортежам. По всей видимости, такой способ хранения был предопределен исторически, поскольку предшествующие иерархические и сетевые СУБД хранили данные в виде «наборов данных», аналогичных кортежам.

Архитектура нашего проекта СУБД была основана на хранении данных по атрибутам, а не по кортежам. Такое решение аргументировалось следующим образом:

- Эффективность хранения, поиска и извлечения информации;
- Простота, компактность и эффективность внутреннего устройств СУБД;
- Концептуальная целостность системы.

Эффективность хранения, поиска и извлечения информации

Хранение данных по атрибутам позволяет хранить только уникальные значения в каждом атрибуте, что обеспечивает компактность базы данных (в те времена это было очень важным условием). Более того, как будет показано ниже, исчезает необходимость хранить внешние ключи с каждым ссылающимся отношением (то есть, реально сохраняются только первичные ключи у главного (master) отношения). Наконец, однородность сохраняемых значений атрибута дает возможность эффективно управлять заполненностью страниц, что сокращает потребности в избыточной оперативной памяти (собственно, для каждого базового типа данных можно заранее или в динамике определить оптимальный размер страницы хранения информации).

Соответственно, для поиска подгружаются только те атрибуты, которые указаны в запросе (в части WHERE…). Остальные атрибуты, не принимающие участие в поиске, не загружаются в память, что экономит время загрузки и открывает возможность быстрого

5 Позже, в середине 90-х годов, я пытался в FIDO общаться с адептами «постреляционных» СУБД, но любую критику в адрес их предмета обожания они отвергали. Тогда такая реакция меня удивила, потом она стала привычной.

сканирования загруженных атрибутов. Однотипные атрибуты очень просто индексировать (настолько просто и быстро, что это можно делать не только по запросу, но и исходя из частоты поиска по данному атрибуту). И, наконец, хранение по атрибутам позволяло без дополнительных затрат параллельно выполнять поиск по каждому атрибуту[6]. Например, если в запросе было указано:

```
... WHERE (“CITY” = 'New York' OR “CITY” = 'London') AND “EVENT_DATE” = '12/24/1967' AND 'TOTAL_COST' > 30000.00 ...
```

то поиск по столбцам “CITY”, “EVENT_DATE” и “TOTAL_COST” может производиться полностью независимо и параллельно. Результаты поиска накладываются друг на друга с помощью логических операций (в данном примере одна операция “OR” и две операции “AND”).

Извлечение информации также не требует загрузки всего кортежа, достаточно загружать только те атрибуты, которые указаны в запросе после ключевого слова “SELECT”. Но перед тем, как будут извлечены необходимые поля, необходимо выполнить слияния (JOIN) таблиц (и, при необходимости, выполнить дополнительную фильтрацию). Слияние таблиц происходит эффективно, поскольку нет необходимости выполнять поиск по внешним ключам. В результате скорость извлечения данных, в большей степени зависит от количества извлекаемых полей и количества полей участвующих в поиске, чем от количества соединяемых таблиц[7].

Простота, компактность и эффективность внутреннего устройства СУБД

Первоначальная архитектура СУБД представлена на Рис. 2

Рис. 2. Архитектура системы управления базами данных

Уровень сервера обеспечивает соединение пользователей с базой данных (или несколькими базами данных). Здесь реализуется поддержка сетевой инфраструктуры. Данный уровень отвечает за регистрацию и авторизацию пользователей на сервере, а так же за регистрацию, создание и удаление баз данных. В расширенной реализации, каж-

6 Не все разработчики соглашались со мной, не все считали параллелизм важной/перспективной особенностью СУБД

7 Позже вышел MS SQL Server, который в первых реализациях очень не любил операцию соединения таблиц. Если в запросе явно или неявно происходило соединение 3-5 таблиц, то запрос выполнялся в несколько раз медленнее, чем при последовательном извлечении тех же данных из тех же таблиц. При соединении большего количества таблиц (до 10-12) сервер баз данных просто «падал», иногда увлекая за собой и операционную систему Windows NT.

дый сервер может являться частью распределенной системы управления базами данных и, соответственно, обладать необходимыми интерфейсами для интеграции с другими серверами.

Уровень баз данных отвечает за поддержание структуры базы данных, назначение/редактирование/удаление прав и ролей пользователей по отношению к объектам базы данных. Проверке прав доступа каждого запроса (чтение/добавление/удаление/изменение). Обеспечивает старт и завершение транзакций, включая распределенные транзакции. На этом уровне ведется журнал, обеспечивается резервное (включая, инкрементальное) копирование и восстановление баз данных; обеспечивается ссылочная целостность. Данный уровень проверяет SQL-запросы на существование объектов и соблюдение правил доступа к ним.

Уровень отношений отвечает за создание, модификацию и удаление таблиц и индексов. Здесь происходит окончательная трансляция SQL-запроса к исполняемому виду; здесь же формируются планы выполнения запроса и выполняется оптимизация. При выполнении запроса уровень отношений поддерживает порядок исполнения запроса, выполняет соединение результатов поиска, обеспечивает формирование выходного потока данных, а также выполняет прием и разбор входного потока данных.

Уровень атрибутов отвечает за размещение данных на внешних носителях и в оперативной памяти; выполняет основные операции: поиск, чтение/извлечение, добавление, изменение и удаление данных.

Компактность и эффективность системы обеспечивается едиными механизмами, используемыми на всех уровнях. На уровне атрибутов хранятся данные представленные, как множество значений одного типа. Но и сами атрибуты представляют собой множество значений типа «атрибут», управляемых с уровня «отношения». В свою очередь, отношения представляют собой множество значений типа «отношение», управляемых с уровня базы данных. И базы данных являются множеством значений типа «база данных» управляемых с уровня сервера баз данных. И, с точки зрения системы, запросы вида “CREATE DATABASE...”, “CREATE TABLE...”, “INSERT INTO...” являются идентичными и выполняемыми одними и теми же частями кода, но при этом они влияют на разные уровни системы. Это единство механизмов является следствием концептуальной целостности системы и само обеспечивает и поддерживает концептуальную целостность системы.

Интерфейсы

Интерфейсы системы можно разделить на две группы. Первая группа обеспечивает взаимодействие с внешней средой. Эти интерфейсы расположены на верхнем и нижнем уровне системы. Вторая группа интерфейсов обеспечивает выполнение SQL-запросов пользователей. Данная группа интерфейсов расположена между названными выше уровнями, то есть, внутри системы. Первая группа интерфейсов рассматриваться в данной работе не будет, поскольку она зависит от внешней среды (структуры предоставляемого ею сервиса), эти могут интерфейсы интересны только специалистам.

SQL-запросы, в зависимости от своего типа, выполняются на разных уровнях системы. Рассмотрим уровни исполнения основных видов запросов

Таблица 1: Уровни исполнения запросов

Запрос	Уровень исполнения
Connect, Disconnect	Сервер баз данных

Create/Delete/Modify user	Сервер баз данных[8]
Create/Drop database	Сервер баз данных
Create/Drop domain/table/view/procedure	База данных
Create role	База данных
Grant/Revoke	База данных
Create exception	База данных
Create sequence	База данных
Start Transaction, Commit Work, Rollback	База данных[9]
Alter table … add/alter/drop	Отношение
Create/Alter/Drop trigger	Отношение
Create index	Отношение
Select, Insert, Update, Delete	Атрибут

Рассматривать все запросы (см. Таблица 1), обременительно, поэтому рассмотрим только запросы уровня атрибута, как наиболее распространенные. Все запросы уровня атрибута, поступающие к СУБД можно представить в виде бинарной иерархии (см. Рис. 3.). Представление типов запросов в виде иерархии удобно по причине того, что позволяет мгновенно определять права доступа любой роли (пользователя) к любому объекту базы данных. Для этого достаточно битовую «маску» запроса наложить на битовую «маску» прав роли по отношению к указанному в запросе объекту базы данных. Предложенная бинарная иерархия не является строгой и не является обязательной, но она эффективна решает задачу авторизации при выполнении запросов.

Окончательный разбор запроса выполняется отношением, которое строит план выполнения запроса. Далее, в соответствии с планом, запрос, разделенный на части, поступает в атрибуты на исполнение. Последовательность выполнения запроса на чтение:

1. поиск (WHERE clause) — необязательно;
2. получение результирующего вектора;
3. выборка (SELECT clause);
4. соединение с выборками из других отношений (JOIN clause) — необязательно;
5. агрегирование (GROUP BY + COUNT, SUM, AVG clause) — необязательно;
6. поиск по агрегированию (HAVING clause) — необязательно;
7. сортировка (ORDER BY clause) — необязательно;
8. формирование выходного потока.

Аналогично строятся последовательности для других видов запросов. Все множество видов обращений отношения к своим атрибутам и образует интерфейсы между уровнем отношений и уровнем атрибутов.

Уровень атрибутов (столбцов)

Атрибуты могут быть разных типов, при этом количество типов атрибутов может

8 Запросы этой группы могут исполняться на уровне сервера баз данных или на уровне базы данных, во втором случае потребуется регистрация пользователей в каждой базе, расположенной на том же сервере.

9 В случае распределенных систем инициатором старта транзакции или инициатором ее завершения может быть уровень сервера базы данных или еще более высокий уровень, объединяющий сервера баз данных в единую распределенную систему.

со временем увеличиваться за счет введения новых типов. Типы атрибутов удобно представлять в виде иерархии, представленной на Рис. 3.

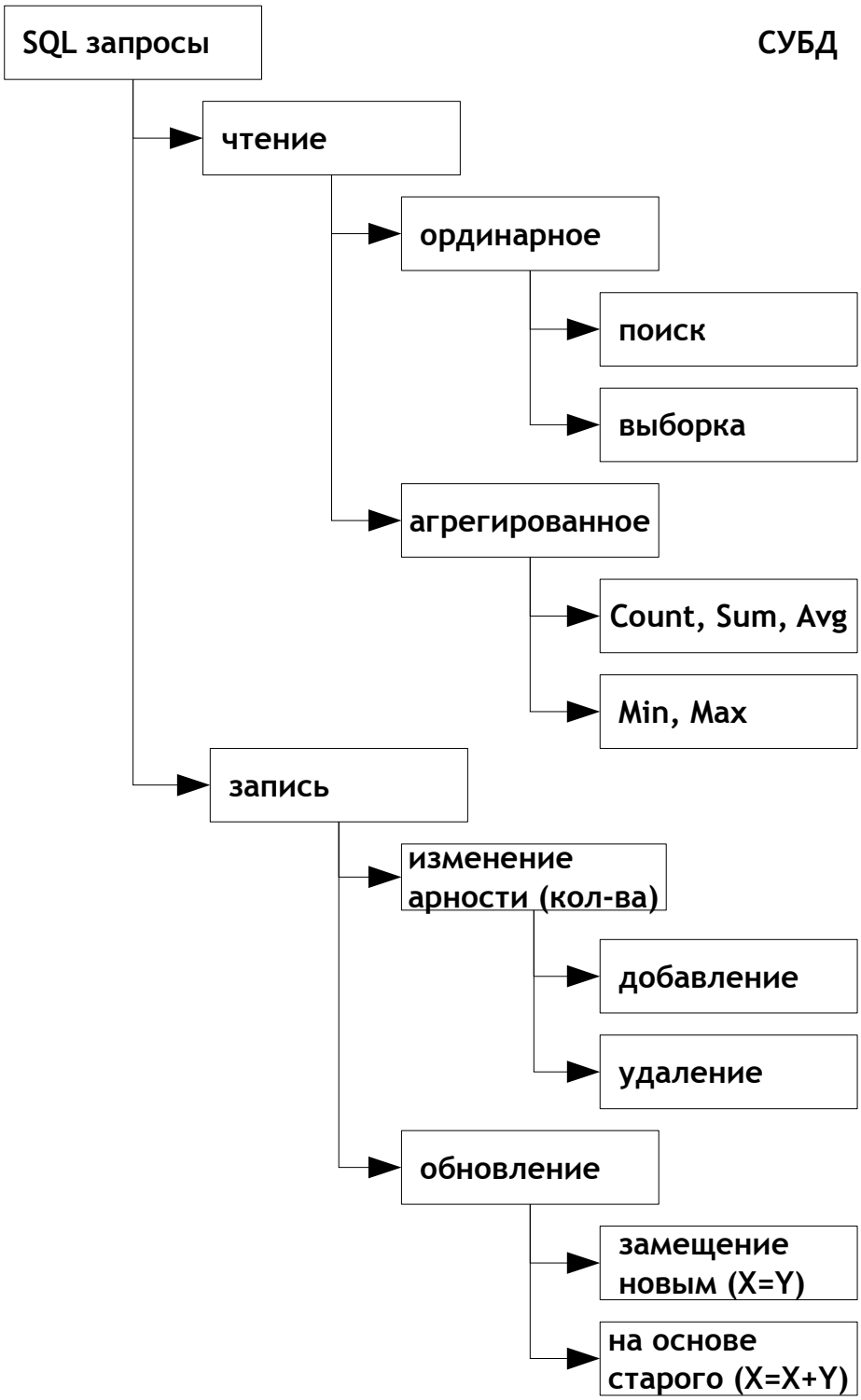

Рис. 3. Бинарная иерархия типов запросов

Базовый (абстрактный) тип атрибута должен соответствовать интерфейсам, представленным ранее, то есть, он должен декларировать поддержку интерфейсов (в простом случае, содержать объявления соответствующих виртуальных методов). Далее, иерархия типов подразделяется на две «ветки»: атрибуты фиксированного размера и атрибуты произвольного размера.

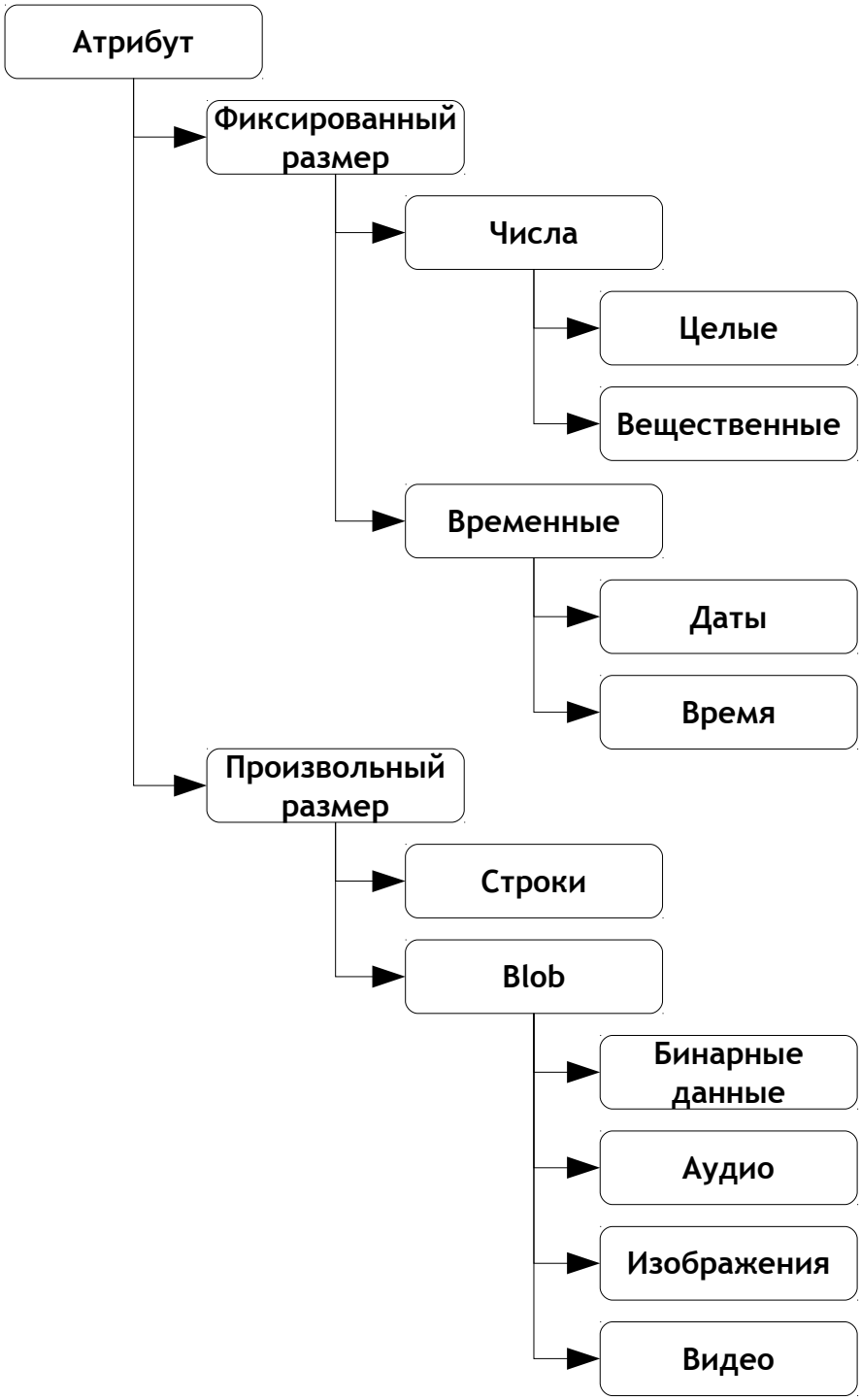

Рис. 4. Типы (классы) атрибутов

С атрибутами фиксированного размера удобно работать, их можно представить в виде массива, и прямо обращаться к любому элементу (произвольный доступ). Скорость обращения к элементам достаточно высока, а объем кода, необходимого для обращения к произвольному элементу, мал. В первых (пробных) реализациях можно было бы рекомендовать именно такую реализацию хранения атрибутов — в виде одномерного массива. В последующем данную реализацию можно заменить на ту, которая обеспечивает эффективную работу с большими объемами данных. Использовать простые реализации желательно, так как они менее подвержены ошибкам, создаются быстрее и дают возможность сосредоточится на отладке логики текущего и смежных с ним уровней. Подмена одной (простой) реализации на другую (эффективную) производится довольно просто (данная

операция не является трудоемкой)[10].

Атрибуты фиксированного типа представлены в стандарте SQL числами и датами (временем) и логическим типом. Логический тип атрибута удобно представлять битовыми векторами (массивами), они, в частности, широко использовались для внутренних задач СУБД. Например, результаты поиска в отношении (таблице) могут быть представлены битовым вектором. Если текущий элемент соответствует критериям поиска, то он получит значение True (1) в битовом векторе, в противном случае текущий бит будет установлен в значение False (0).

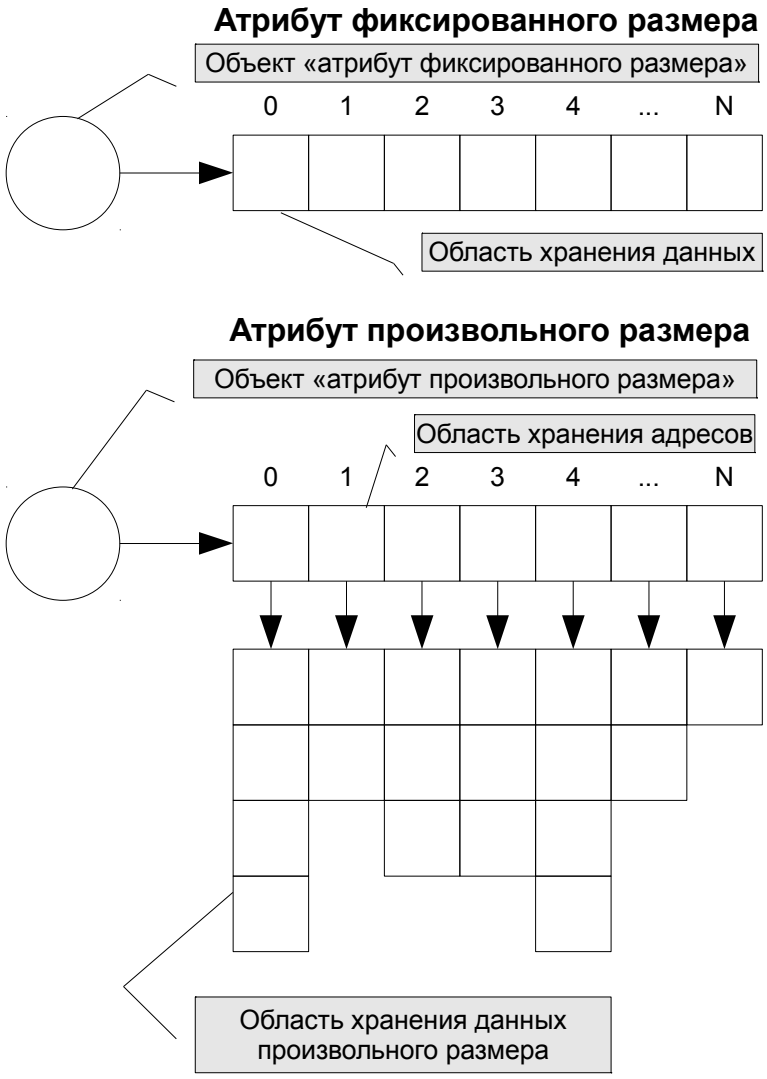

Рис. 5. Атрибуты

Помимо тех операций над атрибутами, которые были представлены ранее, необходимо реализовать дополнительные операции: сравнение, складывание и вычитание, умножение и деление для чисел, сравнения, складывание и вычитание для дат и времени, сравнение и соединение строк, поиск и выделение подстрок и пр.

Задача несколько усложняется, когда от фиксированных типов атрибутов переходим к атрибутам произвольного размера. Действительно, нельзя получить адрес элемента простым умножением его номера на размер, следовательно, происходит замедление

10 Желательно иметь файлы конфигурации, в которых прописываются особенности каждого типа сборки. Это дополнительно облегчает и одновременно документирует разработку проекта.

доступа. Перейти к нужному элементу произвольного размера можно путем просмотра предшествующих элементов или зная его адрес. Первый вариант совершенно непригоден для работы с большим объемом информации. Второй вариант подразумевает отдельное сохранение адресов полей данных. По эффективности доступа второй вариант близок к эффективности доступа у атрибутов фиксированного размера (при условии, что адреса уже загружены в память). Отметим, что все адреса имеют один и тот же размер, а работа с атрибутами фиксированного размера уже была разобрана. Реализовать еще один тип атрибута фиксированного размера (атрибут адресов) большого труда не составляет.

Как видно из Рис. 5 атрибут произвольного размера контролирует две области памяти: область хранения адресов и область хранения данных. Каждая из этих областей может рассматриваться, как область хранения фиксированных данных, поскольку данные произвольного размера, могут быть разбиты на элементарные составляющие, имеющие фиксированный размер. Так, строку можно разбить на символы, а символ в отличие от строки, имеет фиксированный размер. Blob можно рассмотреть, как набор сегментов или байт. В таком случае, атрибут произвольного размера можно рассматривать, как составной, состоящий из двух атрибутов фиксированного размера, имеющих разное количество элементов. Например, пусть атрибут хранит две строки, тогда и количество адресов будет равно двум, а размер области атрибута данных будет равен длине двух строк.

Составной атрибут может включать и большее количество элементов, при этом размеры всех атрибутов фиксированного размера, включенных в составной атрибут, могут быть различны. Рассмотрим следующий пример. Пусть в базе данных будет создано отношение «Адреса», одним из атрибутов которого будет строковый атрибут произвольного размера: «Название населенного пункта». Допустим также, что в базе данных хранятся адреса людей прописанных в конкретном населенном пункте (городе). Тогда у большинства зарегистрированных в базе данных людей название населенного пункта (города) будет одним и тем же. Если хранить данные по кортежам, то придется смириться с тем, что

- строки должны иметь ограничение по длине (CHAR(N); VARCHAR(N); NCHAR(N); NVARCHAR(N));
- одни и те же значения будут храниться многократно (в нашем примере, практически в каждом кортеже будут повторяться одни и те же значения — названия города).

В случае, если данные хранятся по атрибутам, можно не накладывать ограничений на длины строк (как и других типов данных произвольного размера), и можно не хранить дубликаты. Действительно, мы можем однократно сохранить название города в области данных и ссылаться на него многократно в области адресов. При условии, что размер адреса меньше, чем средний размер названия населенного пункта, получим более компактное хранение информации. Ускорится и поиск информации, по той простой причине, что перебирать и сравнивать придется меньше строк. Но, после того, как найдена искомая строка надо определить какие кортежи на нее ссылаются. Значит теперь надо произвести поиск в области хранения адресов, с тем, чтобы выяснить, каким кортежам (строкам таблицы) принадлежат ссылки на найденную строку. Конечно, это негативно скажется на эффективности поиска.

Попробуем решить эту проблему известным путем. Для каждого уникального значения атрибута создадим список кортежей, которые на него ссылаются. Очевидно, что для разных значений, списки будут иметь разную длину. Пусть в нашем примере, на первый населенный пункт ссылаются 5 кортежей, на второй 267689 раз, на третий 1298 раз.

Значит и количество элементов в списках будет соответствовать сумме этих чисел. Таким образом, списки тоже представляют собой атрибуты произвольного размера, состоящие из адресов списка, и, собственно, самого списка (перечня кортежей).

По мере необходимости, можно и дальше развивать атрибуты, добавляя к ним новые элементы[11]. Мало того, можно «научить» систему переходить от одной реализации атрибута к другой в процессе работы, если в этом возникает необходимость. Следует отметить, что наращивание количества элементов в атрибуте может быть связано не только с ускорением доступа, поиска и представления результатов, но и с иными причинами. В частности, чуть позже используя те же механизмы, будут решаться вопросы связанные с оптимизацией связей между отношениями.

Однако стало крайне неудобно говорить об атрибутах, содержащих атрибуты, поскольку постоянно надо уточнять о каком уровне вложения идет речь. Тем более, что логика управления поиском, сохранением и извлечением данных отличается от логики управления вложенными атрибутами. Размышления над этими проблемами привели к пониманию необходимости введения еще одного уровня в систему управления базами данных (Рис. 6).

Рис. 6. Уточненная архитектура СУБД

В уточненной архитектуре уровень пулов отвечает за сохранение, поиск и выдачу данных, а уровень атрибутов связывает единой логикой различные пулы (области хранения в единое целое).

Атрибуты произвольного размера, хранящие только уникальные значения, интересны для последующего изложения, поскольку они допускают, что *количество значений отдельного атрибута может быть меньше мощности отношения* (количества кортежей (строк) в отношении (таблице)). В Таблица 2 представлен пример строкового атрибута «Производитель» в отношении (таблице) «Продукция»

11 Функциональность атрибута не изменяется, не смотря на то, что его внутренняя структура преобразилась кардинально.

Таблица 2: Отношение «Продукция»

Модель	Ревизия	Производитель	Комментарии
S5520HCR	F012	INTEL	
DQ57TML	D103	INTEL	
DG45ID	H221	INTEL	
M4A89TD	KP2	ASUS	
M4A88T-V EVO	YG5	ASUS	
M3A76-CM	UT	ASUS	
GA-H55M	USB3	GIGABYTE	
GA-890XA	UD3	GIGABYTE	

Пул «А» содержит ссылки на строки для соответствующих кортежей. Так кортежи 1-3 ссылаются на строку со значением «INTEL», кортежи 4-6 ссылаются на строку со значением «ASUS», а кортежи 7-8 ссылаются на строку со значением «GIGABYTE». Сами строки хранятся в пуле «Б» (данные).

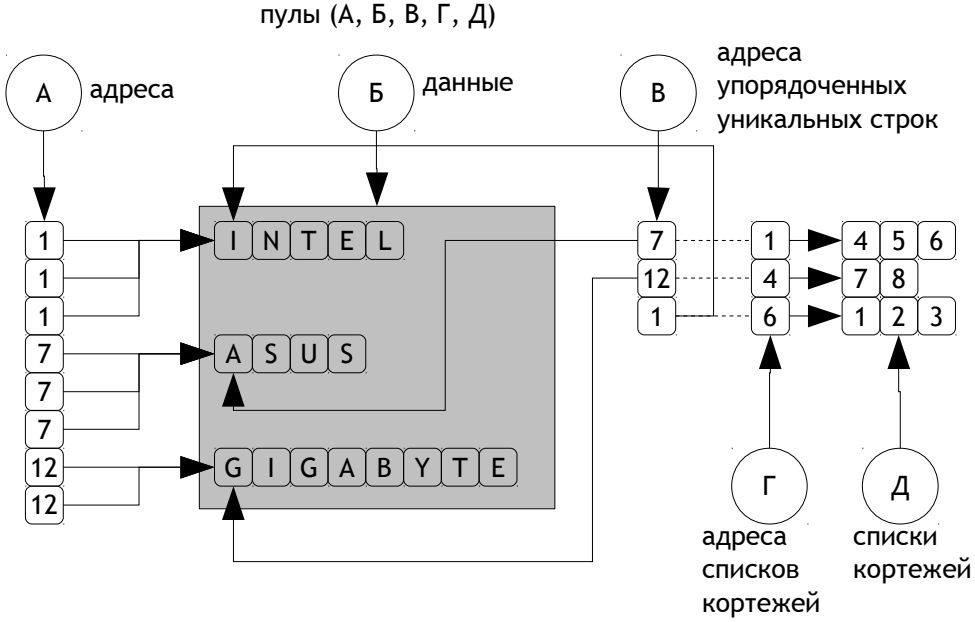

Рис. 7. Структура атрибута «Производитель»

Пул «В» содержит ссылки на уникальные строки, он может быть отсортирован[12] по возрастанию/убыванию значений строк. Количество элементов в этом пуле может быть меньше (если есть дубли) или равно (если нет дублей) количеству кортежей.

Пул «Г» содержит адреса списков кортежей для каждого уникального значения строк. Количество и порядок элементов в этом пуле соответствует количеству и порядку значений в пуле В. Но, если элементы пула В указывают на строки, то элементы пула Г, указывают на начало соответствующего списка.

Пул «Д» содержит списки кортежей. Списки могут быть произвольной длины. Таким образом, совместно пулы Г и Д образуют атрибут произвольного размера.

12 Сортировка значений хранимых в данном пуле позволяет быстрее находить строку с заданным значением, например, при использовании алгоритма бинарного поиска.

Из рассмотренной схемы очевидно, что пулы «Б» и «В» связаны только с хранением данных произвольного размера и не имеют привязки к кортежам. Пулы «А», «Г» и «Д» отвечают за привязку данных к кортежам соответствующего отношения (в данном примере «Продукция»). Очевидно, что те же данные можно аналогичным образом привязать и к другим отношениям, создав пулы «$А_1$», «$Г_1$» и «$Д_1$»; «$А_2$», «$Г_2$» и «$Д_2$» и т. д.. Данный механизм лег в основу создания механизма внешних ключей[13], когда пулы «Б» и «В» проецируются на различные отношения посредством добавления новых пулов «$А_n$», «$Г_n$» и «$Д_n$».

Представленный механизм внешних ключей обеспечивает

1. компактность хранения (данные сохраняются один раз, а не тиражируются в каждом отношении, которое ссылается на данное);
2. высокую скорость поиска, соединения таблиц (отношений) и выборки данных;
3. более высокую надежность хранения, поскольку нет дублирования данных и, следовательно, исчезает возможность их рассогласования;
4. делает простой реализацию каскадных модифицирующих операций в любом варианте (“NO ACTION”, “CASCADE”, “SET DEFAULT” и “SET NULL”), предусмотренном стандартом SQL;
5. позволяет в допустимых пределах выполнять конвертацию атрибута одного типа в другой (например, ANSI-строки в UNICODE, короткие числа в большие, числа и даты в строки, и, наоборот, если строки введены корректно, разумеется);
6. делает бессмысленной практику использования суррогатных ключей при наличии ключей естественных, что, в свою очередь, делает базы данных проще, надежнее и компактнее.

Увеличение количества пулов со списками не более затратно по объемам памяти и скорости, чем добавление индексов, которые по умолчанию создаются на внешних ключах во многих реляционных СУБД.

Разделение логик атрибутов и пулов дает возможность быстрого конструирования новых видов атрибутов, как для стандартных, так и нестандартных (расширенных) типов данных. В свою очередь, конструирование делает возможным и удобным создание шаблонов, за счет чего, можно применять динамичные переходы (run-time) от одной схемы атрибута к другой.

Уровень отношений (таблиц)

Уровень отношений не зависит от того, как реализованы атрибуты. Разработку этого уровня можно начинать тогда, когда реализован хотя бы один простейший тип (или прототип) атрибута.

Отношение (в данном проекте) представляет собой логический набор (список, группу) атрибутов (см. Рис. 8). Логика отношения не зависит ни от количества, ни от типов атрибутов, которые оно включает. Атрибуты могут в динамике добавляться, изменяться (в допустимых пределах) и удаляться.

Атрибуты и их группы могут иметь разную значимость для отношения. Группа атрибутов может образовывать первичный, уникальный или внешний ключ. На атрибуты и их группы могут накладываться дополнительные ограничения. Например, значения атрибута «Дата списания» не могут быть меньше значения атрибута «Дата постановки на учет» для того же кортежа (строки таблицы). Атрибуты и их группы могут образовывать индексы.

13 … с учетом того факта, что первичные и, соответственно, внешние ключи могут быть составными, то есть, содержать несколько атрибутов.

Отношение может иметь триггеры, которые срабатывают на действия связанные с модификацией данных (хотя ничто не препятствует созданию триггеров, срабатывающих на поиск/выборку данных). Триггера могут делиться на те, которые срабатывают до выполнения действия, и на те, которые срабатывают после выполнения действия. Для одного и того же действия можно использовать несколько триггеров, задавая порядок их срабатывания.

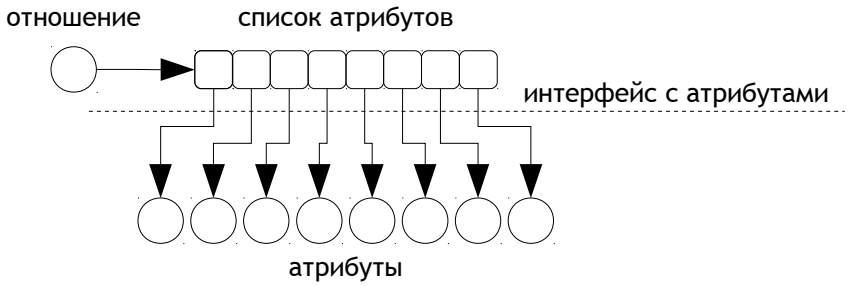

Рис. 8. Структура отношения

Из Рис. 8 видно, что для физического представления отношения, как набора атрибутов, подходит атрибут с фиксированным размером элемента, поскольку ссылки на атрибуты имеют одинаковый (фиксированный) размер. Аналогично сохраняется и другая информация об атрибутах: тип, название, статистическая информация, ограничения и пр.

Для отношения задается следующий порядок выполнения запросов:

1. определение состава атрибутов в запросе;
2. сверка атрибутов из запроса с набором атрибутов отношения;
3. проверка прав доступа к атрибутам;
4. проверка соответствия ограничениям (для модифицирующих запросов);
5. формирование плана исполнения запроса (оптимизация);
6. параметризация поисковых и модифицирующих данных;
7. выполнение запроса атрибутами (для заданных параметров и в соответствии с планом);
8. подготовка и выдача результатов.

При определении состава атрибутов формируется несколько списков, такие как, список выбираемых атрибутов; список атрибутов, по которым производится поиск; список атрибутов для группировки; список атрибутов для упорядочивания с указанием направлений сортировки.

Запрос может выполняться параллельно всеми атрибутами соответствующего списка. Поскольку каждый атрибут хранится независимо от других и результаты также сохраняются независимо друг от друга, то не может возникнуть ситуация пересечения по модифицированным данным в рамках одного запроса. Таким образом, данную систему можно отнести к MPP[14] системам обработки информации.

Рассмотрим пример запроса на выборку или модификацию данных. Запрос содержит следующую WHERE -часть:

… WHERE “MODEL” = 'M4A89TD' AND (“ISSUE_DATE” BETWEEN '07/25/2003' AND '03/31/2009' OR “CUSTOMER” = 'Рога и копыта');

14 MPP (Massively Parallel Processor — процессор с массовым параллелизмом) MPP-системы обладают масштабируемостью близкой к линейной (производительность системы растет пропорционально росту количества вычислительных устройств).

План выполнения поиска[15] будет выглядеть следующим образом:

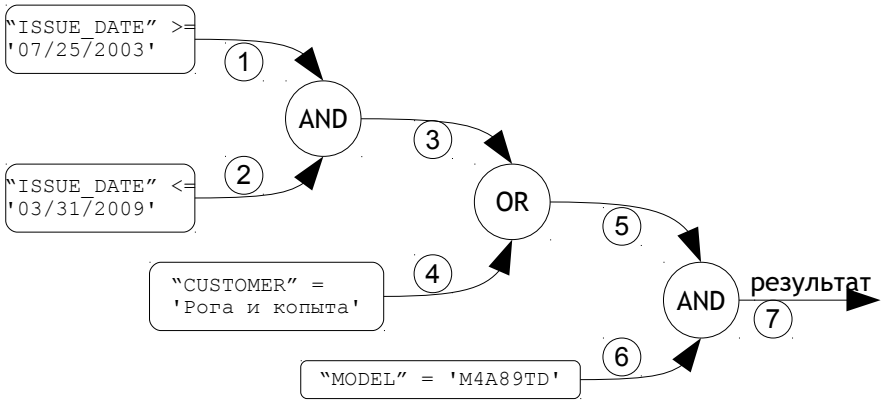

Рис. 9. План поиска

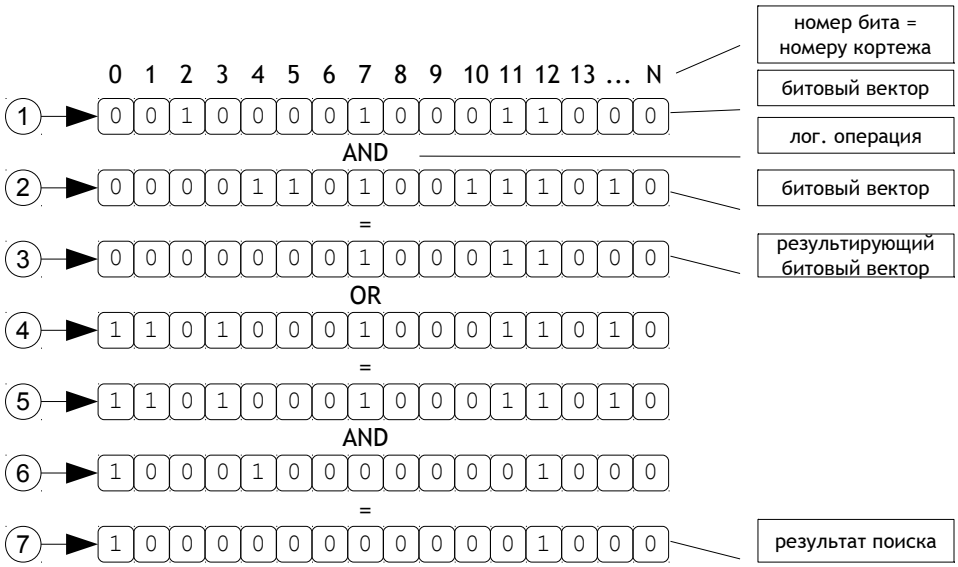

Рис. 10. Получение результата поиска

Результаты запроса формируются в виде битовых векторов, которые потом накладываются друг на друга в соответствии с логической операцией. В данном примере результат сформируется, как показано на Рис. 9 (условиям поиска удовлетворили 0 и 12 кортежи). Наложение битовых векторов является компактной и эффективной вычислительной процедурой[16].

Развитие уровня отношений

Логические View-отношения

View (представление) по своим функциям и возможностям близко к обычному отношению, но View, в отличие от отношения, не хранит данные. View представляет собой селективный SQL-запрос. Использование View вместо обычного запроса к отношениям (таблицам) является очень удобным для решения следующих задач:

15 Это не более, чем иллюстрация, при построении реального плана учитывается статистика, наличие и селективность индексов и другая информация.

16 В проекте была библиотека работы с битовыми векторами, которая позволяла искать, вставлять и удалять битовые «подстроки», сканировать битовый вектор, подсчитывать количество установленных/сброшенных бит, выполнять наложения битовых векторов и т. п.

1. Разделение прав на выборку или изменение столбцов одной или более таблиц;
2. Разделение прав на выборку или изменение строк одной или более таблиц;
3. Отделение логики обращения к данным от логики хранения данных;
4. Модификация данных из «необновляемого»[17] View;

Разделение прав осуществляется за счет запрета пользователю/роли непосредственно читать/модифицировать табличные данные. Но при этом пользователю/роли даются права на чтение/модификацию View, а самому View выдаются необходимые права на таблицы. Теперь пользователь/роль могут читать/менять только то, что возвращает View. Например:

```
CREATE VIEW VIEW_TABLE_A (FLD_V1, FLD_V2, FLD_V3) AS
SELECT TA.“FLD_A”, TA.“FLD_C”, TB.“FLD_Z” FROM “TABLE_A” TA
INNER JOIN “TABLE_B” TB ON TB.“FLD_Y” = TA.“FLD_W” AND TB.“FLD_G” IS NOT NULL
WHERE TA.“FLD_D” = 'secure level 5A' OR TA.“FLD_E” IN (1, 3, 7);
```

В этом примере пользователь/роль видит не все столбцы таблиц TABLE_A и TABLE_B, а только те, которые перечислены FLD_V1 (TA.“FLD_A”), FLD_V2 (TA.“FLD_C”), FLD_V3 (TB.“FLD_Z”) и не все строки, а только те, которые соответствуют условиям поиска. Вся остальная информация остается закрытой для пользователя/роли (к таблицам непосредственного доступа нет, а View показывает ограниченное подмножество данных).

Исторически концепция представлений создавалась для решения задачи разделения логики хранения данных от логики их использования. То есть, представления должны были создавать интерфейс доступа к данным. В таком случае, можно менять схему хранения, оставляя неизменным интерфейс обращения (View). Однако, в реальности существует много ограничений на такое использование представлений. Возможно, поэтому представления для этих целей используют не слишком часто.

Модификация данных посредством «необновляемого» View возможна в случае, если СУБД поддерживает триггеры на View. В таком случае, прямая модификация данных из запроса не происходит, ее выполняют в триггере. Такое решение удобно, когда, например, желательно выполнить вставку данных в несколько таблиц, как в единую таблицу.

СУБД может поддерживать представления двумя способами:

- представления хранятся в виде текстовых описаний (деклараций). При запросе к представлению его текст, по заданным правилам, соединяется с текстом запроса (фактически получаем запрос вида SELECT … FROM SELECT …);
- представления хранятся в препарированном виде, подготовленными к исполнению. На результат выполнения накладываются дополнительные условия из запроса.

Второй случай представляет собой логическое представление одного или более отношений с подготовленными схемами запроса обращения к этим отношениям. Или, говоря иначе, представление — это набор отношений (в том числе, и других представлений) с указанием подзапроса к каждому из отношений.

Как отмечалось выше, отношение — это набор атрибутов, а представление — это логический[18] набор отношений. Следовательно, к представлениям применим подход, изложенный выше.

---

17 «Необновляемое» представление (non-updatable View) – это представление, которое содержит группировку, вычисляемые поля. Обновляемое представление — это такое представление, у которого есть однозначное (1 к 1) соответствие со строками таблицы, и поля представления не должны быть вычисляемыми.

Отдельно следует отметить, что представления не образуют отдельного системного уровня, поскольку интерфейс обращения к отношениям (Tables) и представлениям (View) — один и тот же.

Отношения-Классификаторы

Классификация является удобным способом систематизации и структуризации информации. Существует несколько видов классификаций:

1. Линейная или однофакторная классификация. Для данной классификации характерно разделение фактора на диапазоны/подмножества. Соответственно, попадание значения характеристики сущности в тот или иной диапазон/подмножество, тождественно отнесению сущности к тому или иному классу.
2. Многомерная, табличная или многофакторная классификация. Данная классификация является обобщающей (она включает в себя однофакторную классификацию, как частный случай). Два фактора формируют плоскость, где по одной оси отложены диапазоны/подмножества значений одного фактора, по другой оси — диапазоны/подмножества значений другого фактора. В случае, когда количество факторов произвольно, но конечно, имеем n-мерную классификацию.
3. Иерархическая классификация[19]. Для иерархической классификации характерна неоднородность факторов. В любом месте иерархии могут появляться факторы, которых нет ни у предшествующего узла, ни у других его «потомков». Частным случаем иерархической классификации является семантическая классификация, когда смысл одного понятия вытекает из смысла предшествующего понятия и раскрывается в более частных понятиях.
4. Смешанная классификация. В случае смешанной классификации, каждая область многомерной/линейной классификации может содержать иерархическую классификацию и, наоборот, каждый узел иерархии может содержать многомерную/линейную классификацию.

Потребность в отношениях-классификаторах очевидна и следует из двух положений:

1. Структура базы данных должна моделировать предметную область, и чем более адекватна модель своему оригиналу[20], тем выше качество модели;
2. Во многих предметных областях существуют классификаторы.

Концепция отношений-классификаторов проста, каждый класс реализуется отдельным отношением, где и сохраняется информация. Объекты класса являются кортежами отношения, реализующего данный класс. Запросы могут направляться, как конкретному классу (к конкретному отношению), к классу-«предку» (отношению-«предку»), или к отношению классификатору. Получив запрос, отношение-классификатор (или «предок»), определяет, к каким классам относится данный запрос и, соответственно, транслирует поступивший запрос на эти отношения. Отношения, реализующие классы, сами могут быть отношениями-классификаторами, то есть, содержать подклассы.

18 Наверное, правильнее было бы сказать «виртуальный набор», но принято говорить «логический набор». Так сложилось исторически.

19 Иногда приходится специально акцентировать внимание на том, что не всякая иерархия связана с классификацией и не всякая классификация является иерархической.

20 Адекватность оригиналу оценивается по группе ключевых (для данной предметной области) характеристик

На физическом уровне отношение-классификатор содержит набор таблиц, каждой из которых соответствует некоторое заданное условие (признак принадлежности к классу). При поступлении запроса, отношение-классификатор сравнивает признаки принадлежности с условиями запроса, и перенаправляет запрос к тем таблицам, которые удовлетворяют условию запроса. Подзапросы к разным таблицам могут выполняться параллельно.

После выполнения запроса на каждой из таблиц, результаты сливаются (суммируются) в единый выходной поток, и если необходимо, то группируются (агрегируются) и сортируются, как указано в запросе.

При работе с иерархическими классификаторами надо учитывать тот факт, что структуры полей у отношений в каждом из узлов иерархии могут быть разными. Это является следствием того, что каждый из узлов может иметь уникальные характеристики. Более подробно этот вопрос рассмотрен здесь: http://www.alexus.ru/russian/articles/dbms/oop_rm/index.htm .

Отношения-переключатели

Отношения-переключатели похожи на отношения-классификаторы, они тоже разбивают исходное отношение на множество отдельных отношений по определенному признаку. Есть два основных отличия отношений-переключателей от отношений-классификаторов:

1. отношение-переключатель имеет одинаковый набор атрибутов в всех своих физических таблицах;
2. отношение-переключатель имеет атрибуты переключения, которые используются для разделения данных по таблицам. Эти атрибуты являются обязательными[21] и не сохраняются в кортежах[22].

В качестве иллюстрации рассмотрим пример. Предположим, что необходимо хранить и обрабатывать данные социологического опроса. Предположим также, что обработка данных происходит раздельно для мужчин и женщин. Следовательно, большинство запросов будут содержать условия … WHERE “GENTLE” = 'M' … или … WHERE “GENTLE” = 'Ж' … . Можно заранее раздельно сохранить данные опроса мужчин и женщин, и направлять запросы в зависимости от условия либо, к «мужской», либо к «женской» физическим таблицам. При этом в самих отношениях хранить атрибут “GENTLE” не имеет смысла, так как он во всех кортежах физической таблицы будет иметь одно и то же значение (в «мужской» таблице во всех кортежах будет стоять «М», а в «женской», соответственно, «Ж»).

Теперь рассмотрим более реальный пример. Часто выборка данных происходит за определенный промежуток времени. Очевидно, что было бы целесообразно учитывать этот фактор, при создании физических таблиц базы данных. Можно, например, в качестве переключателя задать диапазон дат с «01/01/20xx» по «01/01/20xx+1». Тогда в каждом новом году будет автоматически создаваться новая таблица, а запросы, содержащие в условиях поиска диапазоны дат будут направляться к соответствующим таблицам. В таком случае, подгружаться в память будут только те таблицы (части отношения-переключателя), которые соответствуют условиям запроса. Если в диапазон, указанный в запросе, попадает более одной физической таблицы, то подзапросы к каждой из физических таблиц могут выполняться независимо и параллельно.

---

21 Ограничение NOT NULL (обязательности) является строгим, то есть, атрибут-переключатель не может быть NULL.

22 Второе ограничение не является строгим, то есть, допускается хранить атрибуты-переключатели.

Не составит труда реализовать аналогичный механизм переключения для любых типов данных. На уровне прикладных разработчиков достаточно расширить SQL описание поля/домена ключевым словом SWITCH [(диапазон [, начальное значение])]. Для числовых типов данных и дат, можно указывать значение диапазона. Для значений, удовлетворяющие данному «диапазону», данные будут помещаться в одну физическую таблицу, при выходе за «диапазон» будет автоматически создаваться новая таблица для данного «диапазона». Значения поля при заданном диапазоне сохраняются в таблице. По умолчанию «диапазон» начинает отсчитываться от нуля, но можно задавать иные начальные точки отсчета, указав соответствующее «начальные значения». Если «диапазон» не задан, то для каждого нового значения будет создана отдельная физическая таблица. В этом случае, значение поля в таблице не хранится. Для строковых полей в качестве «диапазона» может быть указано значение LETTERS [(n)], где n задает, с какой отличающейся буквы формируется новая таблица. По умолчанию n = 1. Например, создадим таблицу MY_TABLE

```
CREATE TABLE MY_TABLE (
…
FLD_SWITCH VARCHAR NOT NULL SWITCH LETTER,
…);
```

И выполнить операторы:

```
INSERT INTO MY_TABLE (…, FLD_SWITCH, …) VALUES (… 'AAA', …);
INSERT INTO MY_TABLE (…, FLD_SWITCH, …) VALUES (… 'ABC', …);
INSERT INTO MY_TABLE (…, FLD_SWITCH, …) VALUES (… 'BCA', …);
```

то первые два оператора поместят данные в одну таблицу (все значения начинаются с «A»), третий оператор поместит данные во вторую таблицу (все значения начинающиеся с «B»). Если же создать таблицу-переключатель так:

```
CREATE TABLE MY_TABLE (
…
FLD_SWITCH VARCHAR NOT NULL SWITCH LETTER(2),
…);
```

И выполнить операторы:

```
INSERT INTO MY_TABLE (…, FLD_SWITCH, …) VALUES (… 'AAA', …);
INSERT INTO MY_TABLE (…, FLD_SWITCH, …) VALUES (… 'AAB', …);
INSERT INTO MY_TABLE (…, FLD_SWITCH, …) VALUES (… 'ABC', …);
INSERT INTO MY_TABLE (…, FLD_SWITCH, …) VALUES (… 'BCA', …);
```

то первые два оператора поместят данные в одну таблицу (все значения начинаются с «AA»), третий оператор поместит данные во вторую таблицу (все значения начинающиеся с «AB», четвертый оператор поместит данные в третью таблицу (все значения, начинающиеся с «BC»).

Если при описании строкового поля указывается его длинна, и его длина равна значению переключателя (n) (например, FLD_SWITCH CHAR(**2**) NOT NULL SWITCH LETTER(**2**)), то для каждого уникального значения этого поля будет создана отдельная таблица, а само значение поля, сохраняться в таблице не будет, поскольку оно одно для всей таблицы (по определению).

Принципиальная схема обработки запросов отношения-переключателя состоит в том, что оно проверяет, какие физические таблицы попадают под действие запроса, на

втором шаге на эти таблицы транслируется запрос. После выполнения запроса, результаты соединяются/суммируются.

Возможно создание и других вариаций на тему «отношение».

Уровень базы данных

База данных содержит набор (список) «отношений». Уровень базы данных не зависит от того, какие виды «отношений» предусмотрены в системе, и не зависит от того, как «отношения» реализованы.

Как было отмечено ранее, задачами данного уровня являются

1. Создание, изменение и удаления объектов базы данных (домены, отношения (таблицы), представления, процедуры) (CREATE/ALTER/DROP DOMAIN/TABLE/VIEW/PROCEDURE);
2. Создание, изменение и удаление ролей (поддержка операторов CREATE/DROP ROLE, GRANT/REVOKE);
3. Проверка прав доступа у запроса, сделанного пользователем/ролью, к объектам базы данных (таблицам[23], представлениям, процедурам);
4. Создание, изменение и удаление последовательностей (CREATE/ALTER/DROP SEQEUNCE);
5. Создание, изменение и удаление исключений (CREATE/ALTER/DROP EXCEPTION);
6. Старт (START TRANSACTION), фиксация (COMMIT WORK) или откат (ROLLBACK) транзакций[24].

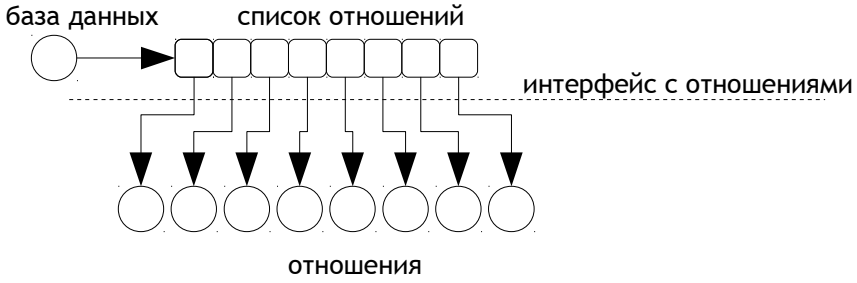

Рис. 11. Структура базы данных

Как видно из Рис. 11 структура базы данных похожа на структуру отношения, но ее элементами являются не атрибуты, а отношения. Список отношений представим списком, который, в свою очередь, реализуется атрибутом с фиксированным размером элемента.

«База данных» взаимодействует с разборщиком запросов (parser). «База данных» отдает разборщику запрос, получая от него готовые к использованию структуры с описа-

23 Права доступа, в соответствии со стандартами SQL-1992-2003, могут распространятся и на атрибуты (столбцы таблиц), но в данном проекте проверка ограничивалась уровнем отношений. Распространить проверку на атрибуты было вполне возможно, но такой задачи в то время не ставилось.

24 Для распределенных транзакций добавляется стадия готовности к завершению (фиксации или откату) транзакции.

нием элементов запроса: типов операций, таблиц, списков полей, с указанием типов полей и значений/параметров[25]. Интерфейс между «базой данных» и разборщиком запросов довольно прост. Структура разобранного запроса представляет собой последовательность, которую должны будут выполнить, как сама «база данных», так и ее отношения (см. раздел «Интерфейсы»).

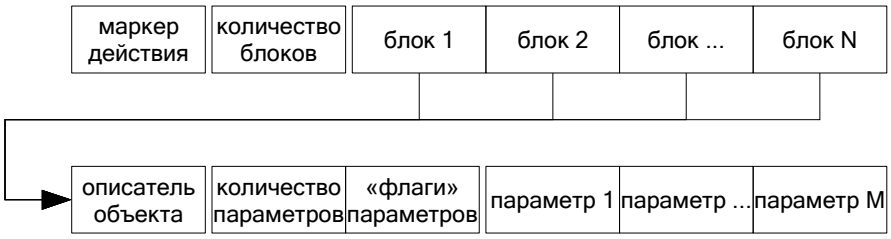

Рис. 12. Структура элементарного действия в запросе

«Маркер действия» описывает определенное действие «поиск», «выборку», «группировку», «сортировку», «добавление», «изменение», «удаление» и пр.. За маркером идет указание количества блоков и сами блоки, описывающие конкретные объекты, выполняющие действия и параметры, необходимые для выполнения действий. В «маркеры действия» могут быть вложены другие «маркеры действия», а в блоки - другие блоки.

После разбора и проверки прав доступа к объектам запроса, структура запроса поступает к указанным отношениям, где производится оптимизация, то есть, определяется порядок выполнения каждой части запроса. Части запроса могут выполняться параллельно и/или последовательно, в зависимости от запроса.

После выполнения запроса «база данных» формирует ответ: готовит выходной поток данных», при необходимости выполняет финальную обработку данных и передает результат в рамках установленной сессии тому, кто послал запрос. Передача может осуществляться частями по требованию или целиком. Тип передачи задает клиентское приложение.

Вопросы, которые не рассмотрены...
<Механизм проверки прав доступа>
<Транзакции>
<Организация ссылочной целостности, чистка пространства БД>
<Журнализация>
<Резервное копирование и «снимки БД»>
<Распределенные БД>
<Организация команды для работы над проектом>
<Организация работ над проектом: планирование версий, распределение работ, контроль, корректировка>

Усов Александр Сергеевич
технический директор ООО «Алексус»
2010-2011 г.г.
г. Екатеринбург

25 Разборщик определяет тип запроса, структурирует сам запрос, разбирая текст, и раскладывая запрос на составные элементы. Разборщик не занимается планированием выполнения запроса и его оптимизацией.