Лекция 15

Концепции хранилищ данных

В основе технологии хранилищ данных лежит идея о том, что базы данных ориентированные на *оперативную обработку транзакций* (Online Transaction Processing — OLTP), и базы данных, предназначенные для делового анализа, используются совершенно по-разному и служат разным целям. Первые — это средство производства, основа каждодневного функционирования предприятия. На производственном предприятии подобные базы данных поддерживают процессы принятия заказов клиентов, учета сырья, складского учета и оплаты продукции, т.е. выполняют главным образом учетные функции. С такими базами данных, как правило, работают клиентские приложения, используемые клерками, производственным персоналом, работниками складов и т.п. В противоположность этому базы данных *интеллектуальных ресурсов предприятия* (business intelligence, BI) (которые Кодд именовал ориентированными на *оперативную аналитическую обработку* (Online Analytical Processing, OLAP)) используются для принятия решений на основе сбора и анализа информации. Их главные пользователи — это менеджеры, служащие планового отдела и отдела маркетинга.

Ключевые отличия аналитических и OLTP-приложений, с точки зрения их взаимодействия с базами данных, перечислены в табл. 21.1. Чтобы лучше понять ее смысл, давайте рассмотрим процесс работы типичных приложений каждого типа. В качестве OLTP-примера возьмем приложение для обработки заказов клиентов. Его обращения к базе данных сводятся обычно к следующему:

- выборка строки из таблицы клиентов для проверки правильности идентификатора клиента;
- проверка лимита кредита клиента;
- выборка строки из таблицы товаров для проверки наличия заказанного товара;
- вставка новой строки в таблицу заказов с данными о новом заказе;
- обновление строки в таблице товаров для уменьшения количества имеющегося в наличии товара.

Таблица 21.1. Сравнение транзакционных и аналитических баз данных

Характеристика базы данных	База данных OLTP	База данных хранилища
Содержимое	Текущие данные	Данные, накопленные за долгий период времени
Структура данных	Структура таблиц соответствует структуре транзакций	Структура таблиц понятна и удобна для написания запросов
Типичный размер таблиц	От тысяч до нескольких миллионов строк	Миллионы и миллиарды строк
Схема доступа	Предопределена для каждого типа обрабатываемых транзакций	Произвольная; зависит от того, какая именно задача стоит перед пользователем в данный момент и какие сведения нужны для ее решения

Окончание табл. 21.1

Характеристика базы данных	База данных OLTP	База данных хранилища
Количество строк, к которым обращается один запрос	Десятки	От тысяч до миллионов
С чем работает приложение	С отдельными строками	С группами строк (итоговые запросы)
Интенсивность обращений к базе данных	Большое количество бизнес-транзакций в минуту или в секунду	Запросы обрабатываются минуты и даже часы
Тип доступа	Выборка, вставка и обновление	Практически 100%-ная выборка
Чем определяется производительность	Время выполнения транзакции	Время выполнения запроса

Рабочая нагрузка на базу данных в таком приложении — это небольшие по объему транзакции, состоящие из простых запросов на выборку, вставку и обновление отдельных строк. Вот примеры таких операций:

- получение цены товара;
- проверка количества имеющегося в наличии товара;
- удаление заказа;
- изменение адреса клиента;
- увеличение лимита кредита клиента.

В противоположность этому типичная транзакция бизнес-анализа (для примера возьмем формирование отчета о заказах) может потребовать выполнения таких операций:

- соединение информации из таблиц заказов, товаров и клиентов;
- вычисление общей стоимости заказов каждого товара каждым клиентом;
- вычисление общего количества единиц каждого заказанного товара;
- сортировка результирующей информации по клиентам.

Рабочая нагрузка на базу данных при получении этой информации представляет собой один долго выполняющийся запрос, требующий выборки большого количества данных. В нашем примере нужно извлечь информацию обо всех заказах, вычислить итоговые и средние значения и сформировать сводную таблицу. Таково большинство запросов из области делового анализа.

- В каких регионах в этом квартале продажа шла наиболее успешно?
- Насколько изменились объемы продаж каждого товара в последнем квартале по сравнению с аналогичным кварталом предыдущего года?
- Каковы тенденции изменения объемов продаж каждого товара?
- Какие клиенты покупают самые ходовые товары?
- Что общего у всех этих клиентов?

Как видите, рабочая нагрузка баз данных, используемых в аналитических и OLTP-приложениях, настолько различна, что очень трудно или даже невозможно подобрать одну СУБД, которая наилучшим образом удовлетворяла бы требованиям приложений обоих типов.

Компоненты хранилища данных

На рис. 21.1 изображена архитектура хранилища данных. Выделим три ее основных компонента.

- **Средства наполнения хранилища** — это набор программ, отвечающий за извлечение данных из корпоративных OLTP-систем (реляционных баз данных, унаследованных баз данных на мэйнфреймах и миникомпьютерах), их обработку и загрузку в хранилище; этот процесс обычно требует предварительной обработки извлекаемых данных, их фильтрации и переформатирования, причем записи загружаются в хранилище не по одной, а целыми пакетами.
- **База данных хранилища** — обычно это реляционная база данных, оптимизированная для хранения огромных объемов данных, их очень быстрой пакетной загрузки и выполнения сложных аналитических запросов.
- **Средства анализа данных** — это программный комплекс, выполняющий статистический и временной анализ, анализ типа "что если" и представление результатов в графической форме.

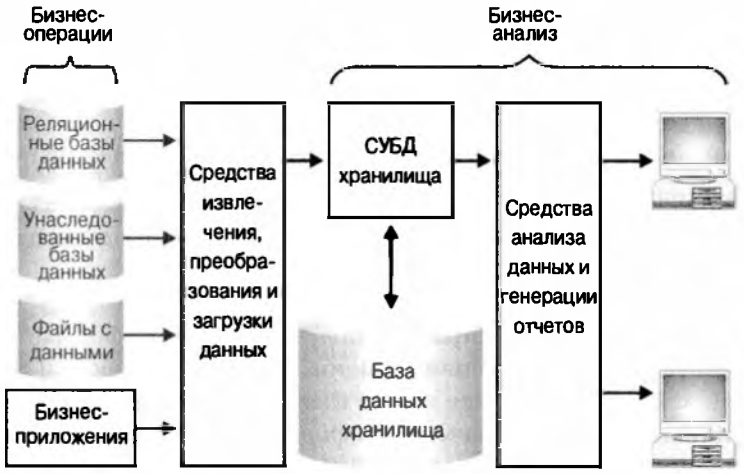

Рис. 21.1. Компоненты хранилища данных

Производители на рынке хранилищ данных сначала сосредоточивали свои усилия в одной из перечисленных областей. Некоторые создавали продукты для наполнения хранилищ, другие сосредоточивались на анализе данных. Кое-кто пытался охватить обе области, но в обеих областях продолжали работу независимые производители программного обеспечения, включая нескольких, доход которых достигал 100 миллионов долларов.

Вначале, когда рынок хранилищ данных только начинал свое развитие, были попытки разработок специализированных баз данных для хранилищ. Со временем этот вопрос заинтересовал и ведущих производителей СУБД. Некоторые из них разработали собственные специализированные базы данных, другие просто перекупили более мелкие компании с их разработками. Сегодня показанные на рис. 21.1 компоненты почти всегда представляют собой специализированные основанные на SQL СУБД, поставляемые одним из ведущих производителей баз данных уровня предприятия.

Эволюция хранилищ данных

Поначалу идея хранилища данных заключалась в создании огромного собрания всех данных предприятия, накопленных за весь период его работы. К такому хранилищу можно было бы адресовать практически любые возможные запросы, касающиеся истории деловой жизни компании. Многие компании пытались создать подобные хранилища, но мало у кого это получилось. На практике оказалось, что огромное хранилище масштаба предприятия создать не только трудно, но и дорого, и, что гораздо важнее, — его довольно неудобно использовать.

Поэтому со временем акцент сместился на хранилища данных для отдельных аспектов бизнеса; размещаемая в них информация носила конкретную практическую направленность и поддавалась более глубокому и эффективному анализу. Такие хранилища, меньшие, но все еще очень объемные, стали называть *центрами данных* (data marts). Именно управление распределенными центрами данных предприятия стало в последнее время основным объектом внимания производителей корпоративных СУБД. В частности, особое внимание уделяется выборке и форматированию данных в ситуациях, когда несколько центров данных извлекают информацию из одной и той же производственной базы данных. Это требует координации их действий, чтобы не получилось возврата к огромным централизованным хранилищам. Еще один подход заключается в том, чтобы оставить данные на месте, в базах данных OLTP, и формировать центры данных по запросам с помощью специализированного инструментария промежуточного уровня, который заставляет данные в нескольких базах данных иметь вид хранящихся в одной огромной интегрированной базе данных, которую можно рассматривать как виртуальное хранилище данных. В такой архитектуре, известной как информационная интеграция предприятия (enterprise information integration, EII), инструментарий промежуточного уровня тиражирует каждый запрос среди всех поддерживаемых физических баз данных и собирает результаты перед тем, как вернуть их пользователю, пославшему исходный запрос.

Хранилища и центры данных используются в самых разных отраслях. Но, пожалуй, наиболее широко (и агрессивно) они применяются в тех промышленно-финансовых сферах, где информация о тенденциях бизнеса служит основой для принятия решений, приводящих к значительной экономии средств или приносящих большую прибыль. Сюда входит следующее.

- **Крупные производства** — анализ тенденций в области сбыта, в частности сезонных колебаний объемов продаж, может помочь компании эф-

фективнее спланировать производство, разгрузить склады и сэкономить деньги для других целей.

- **Коммерция** — анализ эффективности мероприятий, направленных на увеличение сбыта (реклама, доставка товаров на дом и т.п.) и изучение демографических факторов, помогает выявить наиболее эффективные способы привлечения потенциальных покупателей и сэкономить на таких мероприятиях миллионы долларов.
- **Телекоммуникации** — анализ схемы звонков клиентов может помочь в создании более привлекательных комплексов цен и услуг и, возможно, привлечь новых клиентов из числа тех, которые пользуются услугами других компаний.
- **Авиакомпании** — анализ схемы перемещений пассажиров является основой планирования рейсов, тарифов и объемов перевозок с целью максимального увеличения прибылей компании.
- **Финансовые структуры** — анализ факторов, связанных с получением и погашением кредитов клиентами, и их сравнение с данными прошлых лет позволяют делать более точные заключения о кредитоспособности клиентов.

Архитектура баз данных для хранилищ

Структура базы данных для хранилища обычно разрабатывается таким образом, чтобы максимально облегчить анализ информации; ведь это основная функция хранилища. Данные должно быть удобно "раскладывать" по разным направлениям (называемым *измерениями*). Например, сегодня пользователь хочет просмотреть сводку продаж товаров по регионам, чтобы сравнить объемы продаж в разных областях страны. Завтра тому же пользователю понадобится картина изменения объемов продаж по регионам в течение определенного периода — ему нужно будет узнать, в каких регионах объемы продаж растут, а в каких, наоборот, сокращаются и какова динамика этих изменений. Структура базы данных должна обеспечивать проведение подобных типов анализа, позволяя выделять данные, соответствующие заданному набору измерений.

Кубы фактов

В большинстве случаев информация в базе данных хранилища может быть представлена в виде N-мерного куба (N-куба) фактов, отражающих деловую активность компании в течение определенного времени. Простейший трехмерный куб данных о продажах изображен на рис. 21.2. Каждая его ячейка представляет один факт — объем продаж в долларах. Вдоль одной грани куба (одного измерения) располагаются месяцы, в течение которых выполнялись отражаемые кубом продажи. Второе измерение составляют категории товаров, а третье — регионы продаж. В каждой ячейке содержится объем продаж для соответствующей комбинации значений по всем трем измерениям. Например, значение $50475 в левой верхней ячейке — это объем продаж *одежды* в *январе* в *восточном* регионе.

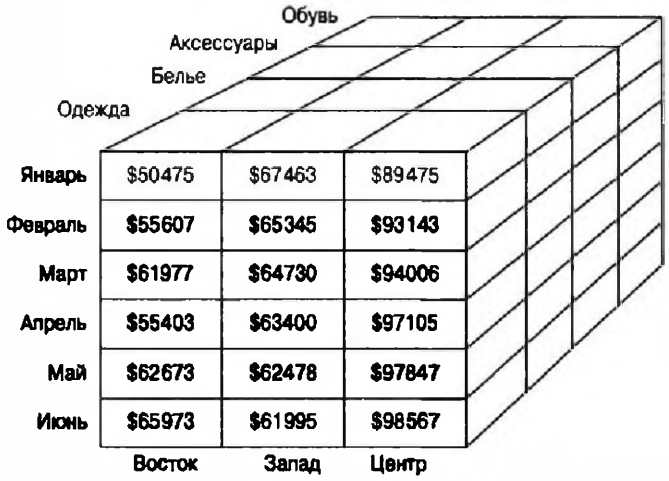

Рис. 21.2. Трехмерное представление данных о продажах

На рис. 21.2 показан трехмерный куб, но в реальных приложениях используются более сложные кубы с десятками и более измерений. Впрочем, хотя десятимерный гиперкуб трудно визуализировать (если вообще возможно), принципы его организации те же, что и у трехмерного. Каждое измерение представляет некоторую переменную величину, по которой ведется анализ. Каждой комбинации значений всех измерений соответствует одно значение некоторого параметра — факт, который обычно является зафиксированным в системе результатом деятельности компании.

Для иллюстрации структуры базы данных, обычно используемой в хранилищах данных, давайте разберем работу типичного аналитического приложения на простом примере. Возьмем коммерческую компанию, которая продает несколько категорий товаров разных производителей и имеет несколько сотен клиентов в разных регионах страны. Руководству компании требуется провести анализ данных о динамике продаж по пяти измерениям, анализ должен осветить имеющиеся тенденции и помочь выявить возможности увеличения сбыта. Вот эти пять измерений, определяющих пятимерный гиперкуб данных, фактами которого являются объемы продаж.

- **Категория**. Категория продаваемого товара (обувь, аксессуары, белье, одежда и т.д.). В хранилище содержатся данные о примерно двух десятках категорий товаров.
- **Производитель**. Производитель конкретного товара. Компания может торговать товарами 50 разных производителей.
- **Клиент**. Покупатель товаров. У компании несколько сотен клиентов. Крупнейшие из них заказывают товары в центральном офисе и обслуживаются одними и теми же служащими. Другие покупают товары в локальных отделениях компании и обслуживаются их сотрудниками.
- **Регион**. Регион страны, в котором продаются товары. Одни клиенты компании всегда покупают товар в одном и том же регионе, другие могут делать это в нескольких регионах.

- **Месяц.** Месяц, в котором были проданы товары. Для сравнительного анализа руководство компании решило ограничиться хранением данных за последние 36 месяцев (3 года).

Каждое из этих пяти измерений относительно независимо от остальных. Объемы продаж по конкретному клиенту могут быть рассредоточены по нескольким регионам. Товары конкретной категории могут поставляться одним или несколькими производителями. Фактом в каждой ячейке пятимерного куба является объем продаж для конкретной комбинации значений пяти измерений. Всего куб содержит более 35 миллионов ячеек (24 категории × 50 производителей × 300 клиентов × × 3 региона × 36 месяцев).

Схема звезды

Для большинства хранилищ данных самым эффективным способом моделирования N-мерного куба фактов является *схема звезды*. На рис. 21.3 показано, как выглядит такая схема для хранилища данных коммерческой компании, описанного в предыдущем разделе. Каждое измерение куба представлено таблицей его значений. На рис. 21.3 таких таблиц пять: `CATEGORIES` (категории), `SUPPLIERS` (производители), `CUSTOMERS` (клиенты), `REGIONS` (регионы) и `MONTHS` (месяцы). Для каждого значения измерения в таблице имеется отдельная строка. Например, в таблице `MONTHS` 36 строк, по одной для каждого месяца всего периода, за который анализируются данные о продажах. А в таблице `REGIONS` три региона представлены тремя строками.

Таблицы измерений в этой схеме часто содержат столбцы с описательной текстовой информацией или другими атрибутами, связанными с конкретным измерением (такими, как имя контактного лица из фирмы-производителя, адрес и номер телефона клиента или сроки контракта). Эти столбцы могут выводиться в отчетах, генерируемых аналитическим приложением. У таблицы измерения всегда имеется первичный ключ, индексирующий значения этого измерения. В таблице измерений всегда есть один или несколько столбцов, содержащих естественные идентификаторы измерений, такие как код региона, месяц и год или размер одежды. Однако эти естественные идентификаторы редко используются в качестве первичных ключей таблиц измерений из-за того, что со временем естественные идентификаторы могут изменяться. Это явление известно как *медленно изменяющиеся измерения*. Чтобы избежать изменений значений первичных ключей, обычно в их качестве в таблицах фактов используются числа, не имеющие реального смысла и известные как *суррогатные ключи*. Суррогатные ключи упрощают внешние ключи, поскольку они состоят из одного столбца. Без такого суррогатного ключа в таблице измерения `MONTHS` ее внешний ключ в таблице фактов состоял бы из двух столбцов — месяца и года.

В хранилище на рис. 21.3 суррогатные ключи используются во всех таблицах. Однако следует обратить внимание на то, что в таблицы измерений включены и естественные идентификаторы — например, имена регионов в таблице `REGIONS` ("Запад", "Восток" и т.д.) или названия категорий в таблице `CATEGORIES` ("Одежда", "Обувь" и т.д.).

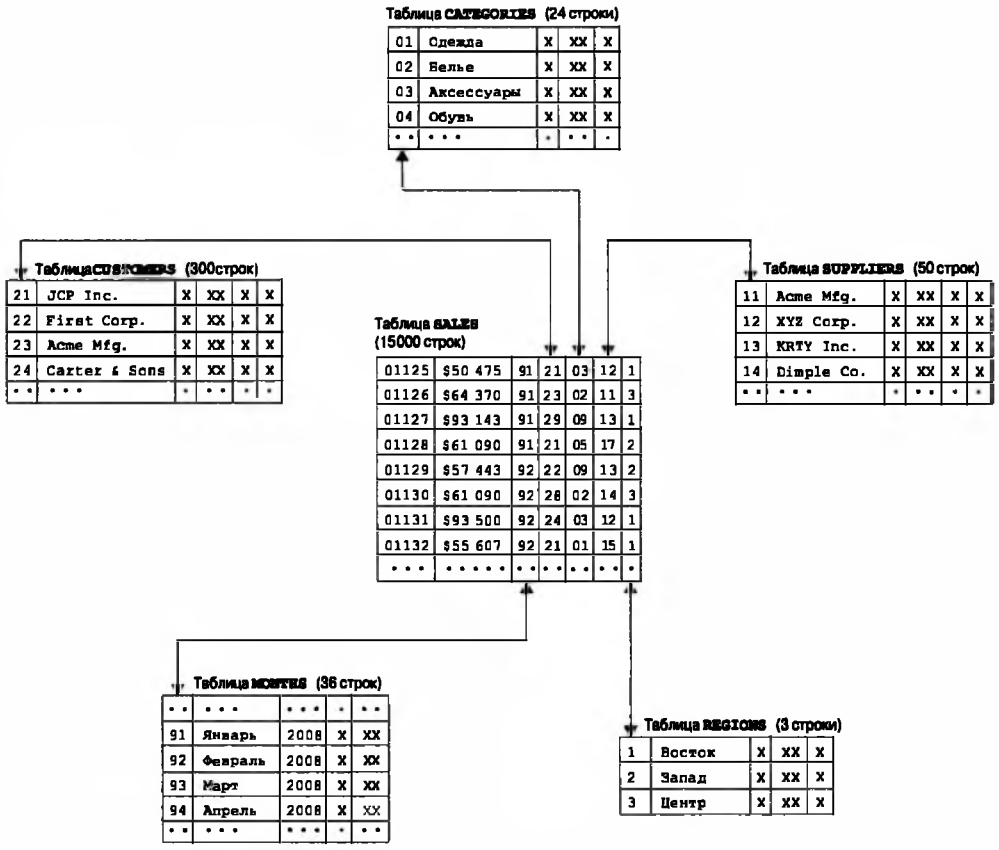

Рис. 21.3. Схема звезды для хранилища данных коммерческой компании

Самой большой таблицей в базе данных является *таблица фактов*. Она — центр схемы. На рис. 21.3 это таблица продаж SALES. Таблица фактов содержит столбец со значениями, которые находятся в ячейках N-мерного куба (см. рис. 21.2). Кроме него, в таблице фактов имеются столбцы с внешними ключами для всех таблиц измерений. Внешние ключи связывают строки с соответствующими строками таблиц измерений. В нашем примере таких внешних ключей пять. В данной структуре каждая строка представляет данные одной ячейки N-куба. Однако простой куб в состоянии показать только три измерения. Чтобы справиться с дополнительными измерениями, следует визуализировать дополнительные кубы. N-куб на рис. 21.2 представляет измерения REGIONS, CATEGORIES и MONTHS. Чтобы представить измерение SUPPLIERS, N-куб следует повторить 50 раз, по одному разу для каждого возможного производителя. Далее, для представления измерения CUSTOMERS требуется эти 50 кубов продублировать по 300 раз — по одному разу для каждого возможного клиента (итого, 15000 кубов). К счастью, некоторые многомерные СУБД в состоянии представлять кубы для анализа без физического хранения каждого куба.

В таблице фактов обычно немного столбцов, но зато очень много строк — сотни тысяч и даже миллионы; это вполне обычный размер хранилищ данных в промышленных и коммерческих компаниях. Столбец фактов почти всегда содержит

числовые значения, например денежные суммы, количество проданного товара и т.п. Подавляющее большинство отчетов, генерируемых аналитическими приложениями, содержит только итоговые данные — суммы, средние, максимальные и минимальные значения, процентные отношения, — получаемые путем арифметических операций над числовыми значениями из столбца фактов.

Причина, по которой схема на рис. 21.3 названа звездой, достаточно очевидна. Концы звезды образуются таблицами измерений, а их связи с таблицей фактов, расположенной в центре, образуют лучи. При такой структуре базы данных большинство запросов из области делового анализа объединяет центральную таблицу фактов с одной или несколькими таблицами измерений. Вот несколько примеров.

Показать итоговые объемы продаж одежды за январь 2008 года по регионам.

```
SELECT SUM(SALES_AMOUNT), REGION
  FROM SALES, REGIONS
 WHERE MONTH = 'Январь'
   AND YEAR  = 2008
   AND PROD_TYPE = 'Одежда'
   AND SALES.REGION = REGIONS.REGION
 ORDER BY REGION;
```

Показать средние объемы продаж товаров каждого производителя по клиентам и по месяцам.

```
SELECT AVG(SALES_AMOUNT), CUST_NAME, SUPP_NAME, MONTH, YEAR
  FROM SALES, CUSTOMERS, SUPPLIERS
 WHERE SALES.CUST_CODE = CUSTOMERS.CUST_CODE
       SALES.SUPP_CODE = SUPPLIERS.SUPP_CODE
 GROUP BY CUST_NAME, SUPP_NAME, MONTH, YEAR
 ORDER BY CUST_NAME, SUPP_NAME, MONTH, YEAR;
```

Многоуровневые измерения

В схеме звезды, изображенной на рис. 21.3, каждое измерение имеет только один уровень. На практике же довольно распространена более сложная структура базы данных, в которой измерения могут иметь по нескольку уровней.

- Данные о продажах могут накапливаться отдельно для каждого офиса. Офисы располагаются в районах, а районы в регионах.
- Данные о продажах накапливаются по месяцам, но анализировать их полезно не только по месяцам, но и по кварталам.
- Данные о продажах накапливаются по отдельным продуктам, а каждый продукт ассоциирован с конкретным производителем.

Подобные многоуровневые измерения усложняют звездообразную схему и на практике возникающую проблему решают по-разному.

- **Дополнительные данные в таблицах измерений**. Таблица измерения `REGIONS`, связанная с географическим местоположением, может содержать информацию об отдельных офисах и при этом иметь столбцы, указывающие, к какому району и региону относится каждый офис. Агреги-

рованные данные, касающиеся районов и регионов, могут быть получены с помощью итоговых запросов, объединяющих таблицу фактов с таблицей измерения и группирующих данные по столбцам района и региона. Этот подход концептуально довольно прост, но не всегда позволяет получить нужные данные за один раз: чаще всего требуется проводить вычисления в несколько этапов, запрос за запросом. Из-за этого на формирование итоговых данных может уходить слишком много времени.

- **Многоуровневые таблицы измерений**. Таблица географического измерения может быть расширена, чтобы в ней были отдельные строки для офисов, районов и регионов. Строки, содержащие суммарные данные для измерений более высокого уровня, добавляются в таблицу фактов при ее обновлении. Это решает проблему производительности запросов, поскольку промежуточные итоговые данные уже вычислены. Однако формулировать запросы становится сложнее, поскольку данные о любой сделке дублируются и присутствуют в таблице фактов в трех строках: для офиса, района и региона. Поэтому вычисление любых статистических данных требует большой аккуратности. Обычно при такой схеме базы данных в таблицу фактов включают столбец уровня, указывающий, к какому уровню принадлежит стоящее в строке значение, и каждый запрос, вычисляющий итоги и другие статистические данные, должен содержать условие для отбора записей только одного конкретного уровня.
- **Предварительно вычисленные итоги в таблицах измерений**. Вместо усложнения таблицы фактов, можно помещать предварительно вычисленные итоговые данные в таблицы измерений. Например, итоговый объем продаж по району может храниться в строке этого района в таблице географического измерения. Это решает проблему дублирования фактов, присущую предыдущему решению, однако годится только для очень простых случаев. Готовые итоговые суммы по районам не помогут получить сводки продаж отдельных товаров по районам или проследить изменение динамики продаж в районах по месяцам, а бесконечное увеличение сложности таблицы географического измерения — не выход.
- **Таблицы фактов для разных уровней**. Вместо усложнения единой таблицы фактов, этот подход предполагает создание нескольких таблиц фактов для разных уровней группировки данных. Для поддержки межуровневых запросов (например, о продажах по районам и месяцам) нужна специальная таблица фактов, в которой данные просуммированы для этих уровней измерений. Результирующая схема таблиц измерений и фактов имеет множество перекрестных связей и напоминает снежинку, поэтому схемы баз данных этого типа обычно так и называют — *схема снежинки*. Этот подход решает проблему производительности и устраняет дублирование данных в таблицах фактов, но может существенно усложнить структуру баз данных хранилищ, поскольку для каждого типа возможных запросов нужна своя таблица фактов. Кроме того, многие популярные инструменты анализа данных не умеют работать с такими схемами.

На практике поиск оптимальной схемы и архитектуры конкретного хранилища представляет собой сложное комплексное решение, учитывающее особенности фактов и измерений, типичные запросы и многое другое. Многие компании прибегают для этого к услугам сторонних консультантов.

Расширения SQL для хранилищ данных

Теоретически реляционная база данных звездообразной структуры обеспечивает хороший фундамент для выполнения запросов из области делового анализа. Возможность легко связывать представленную в базе данных информацию на основе значений данных как нельзя лучше подходит для произвольных, неструктурированных запросов, свойственных аналитическим приложениям. Однако между типичными аналитическими запросами и возможностями базового языка SQL имеются серьезные нестыковки, некоторые из которых перечислены ниже.

- **Сортировка данных.** Многие аналитические запросы явно или неявно требуют предварительной сортировки данных. Вам наверняка не раз встречались такие запросы, как "первые десять процентов", "первая десятка", "самые низкопроизводительные" и т.п. Однако ориентированный на работу со множествами язык SQL оперирует неотсортированными наборами записей. Единственным средством сортировки данных в нем является предложение `ORDER BY` в инструкции `SELECT`, причем сортировка выполняется в самом конце процесса, когда данные уже отобраны и обработаны.
- **Хронологические последовательности.** Многие запросы к хранилищам данных предназначены для анализа изменения некоторых показателей во времени: они сравнивают результаты этого года с результатами прошлого, результаты этого месяца с результатами того же месяца в прошлом году, показывают динамику роста годовых показателей в течение ряда лет и т.п. Однако очень трудно, а иногда и просто невозможно получить сравнительные данные за разные периоды времени в одной строке, возвращаемой стандартной инструкцией SQL. В общем случае это зависит от структуры базы данных.
- **Сравнение с итоговыми данными.** Многие аналитические запросы сравнивают значения отдельных элементов (например, объемы продаж отдельных офисов) с итоговыми данными (например, объемами продаж по регионам). Такой запрос трудно выразить на стандартном диалекте SQL. А если вам нужен сводный отчет классического формата — с детальными данными, промежуточными и общими итогами, то его с помощью SQL вообще нельзя сгенерировать, поскольку структура всех строк в таблице результатов запроса должна быть одинаковой.

Пытаясь решить все эти проблемы, производители СУБД для хранилищ данных обычно расширяют в своих продуктах возможности языка SQL. Например, в СУБД компании Red Brick (позже купленная компанией Informix, в свою очередь, купленной IBM), которая была одним из пионеров рынка хранилищ данных, в язык RISQL (Red Brick Intelligent SQL) включены следующие расширения:

- **диапазоны** — позволяют формулировать запросы вида "отобрать первые десять записей";
- **перемещение итогов и средних** — используется для хронологического анализа, требующего предварительной обработки исходных данных;
- **расчет текущих итогов и средних** — позволяет получать данные по отдельным месяцам плюс годовой итог на текущую дату и выполнять другие подобные запросы;
- **сравнительные коэффициенты** — позволяют создавать запросы, выражающие отношение отдельных значений к общим и промежуточным итогам без использования сложных подчиненных запросов;
- **декодирование** — упрощает замену кодов из таблиц измерений понятными именами (например, замену кодов товаров их названиями);
- **промежуточные итоги** — позволяют получать результаты запросов, в которых объединена детализированная и итоговая информация, причем с несколькими уровнями итогов.

Другие разработчики СУБД для хранилищ данных используют аналогичные расширения SQL в своих реализациях или встраивают такие возможности в свои инструменты для анализа. Как и в случае с другими расширениями SQL, их концептуальные возможности, предлагаемые различными разработчиками, сходны, а реализация зачастую совершенно разная.

Производительность хранилищ данных

Производительность хранилища данных — это одна из его важнейших характеристик. Если запросы, связанные с деловым анализом, выполняются слишком долго, пользователи не станут использовать хранилище для получения ответов на вопросы, возникающие в ходе принятия решений. Если данные слишком долго загружаются в хранилище, информационная служба предприятия откажется от их частого обновления, и отсутствие актуальной информации снизит полезность хранилища. Поэтому важнейшей задачей планирования хранилища является достижение оптимального соотношения между быстротой выполнения запросов и быстротой загрузки свежих данных.

Скорость загрузки данных

Загрузка в хранилище очередной порции данных — дело не быстрое. Этот процесс может занимать часы, а для больших хранилищ — даже дни. Обычно он состоит из следующих операций.

- **Извлечение данных**. Как правило, загружаемые в хранилище данные поступают из нескольких различных источников. Среди них могут быть реляционные базы данных OLTP-приложений.

- **Очистка данных**. Поступающие данные часто бывают "грязными" в том смысле, что содержат значительные ошибки. В частности, старые OLTP-системы могут не выполнять строгого контроля целостности, допуская, например, ввод записей с неверными идентификаторами клиентов или товаров. Поэтому в процессе наполнения хранилища обычно производится проверка целостности данных.
- **Перекрестная проверка данных**. Во многих компаниях для поддержки тех или иных деловых операций используются OLTP-приложения, написанные в разное время и не интегрированные в единое целое. Изменение данных в одной системе (например, добавление новых товаров в приложении для обработки заказов) может не отражаться автоматически в других приложениях (например, в системе складского учета) или отражаться, но с большими задержками. Когда данные из таких разобщенных систем поступают в хранилище, их необходимо проверять на согласованность.
- **Переформатирование данных**. Формат поступающих данных может сильно отличаться от формата, используемого в базе данных хранилища. Например, если текстовые данные переносятся с мэйнфреймов, их нужно транслировать из кодировки EBCDIC в ASCII. Еще один распространенный источник расхождений — форматы дат и времени. Кроме этих очевидных отличий, могут быть и более серьезные: информация из одной таблицы OLTP-источника может разделяться на части, помещаемые в несколько таблиц хранилища, и, наоборот, информация из нескольких таблиц источника может "сливаться" для помещения в одну таблицу хранилища.
- **Вставка/обновление данных**. После предварительной обработки выполняется помещение данных в хранилище. Обычно это специализированная операция, выполняемая в пакетном режиме без транзакций, но зато с использованием специальных средств восстановления в случае сбоев. Ее скорость должна быть очень высокой — обычно до сотен мегабайтов в час.
- **Создание/обновление индексов**. Специализированные индексы, используемые программным обеспечением хранилища, должны быть модифицированы в соответствии с обновленным содержимым базы данных. Как и в случае вставки/обновления данных, эта операция может потребовать особой обработки. В одних ситуациях быстрее и проще создать весь индекс заново, чем модифицировать его по мере поступления строк, в других ситуациях более приемлем второй вариант.

Все эти задачи обычно выполняются в пакетном режиме специализированными утилитами, отвечающими за наполнение хранилища. Когда идет загрузка, выполнение пользовательских запросов к базе данных запрещается, чтобы работа могла выполняться с максимальной скоростью. Несмотря на предельную оптимизацию всех операций, время загрузки информации в хранилище по мере увеличения объема его базы данных постепенно растет. Поэтому планирование соотношения скорости загрузки и скорости выполнения запросов должно проводиться на перспективу. Хранилища с большим количеством индексов или заранее вычисляемыми итоговыми значениями могут обеспечивать очень высокую производитель-

ность запросов, но загрузка данных в них требует слишком много времени. В хранилища более простой структуры данные загружаются быстрее, но запросы некоторых типов могут выполняться недопустимо долго. Поэтому перед администратором хранилища стоит задача нахождения оптимального баланса двух составляющих производительности.

Производительность запросов

Разработчики СУБД для хранилищ данных вкладывают огромные средства в оптимизацию своих продуктов. И результаты их усилий по-настоящему впечатляют. Однако наряду с производительностью растут объемы и сложность хранилищ, поэтому конечный пользователь зачастую может вообще не заметить никаких изменений.

Для повышения производительности запросов, связанных с деловым анализом, используется несколько технологий.

- **Специализированные схемы индексации.** Типичные аналитические запросы извлекают из хранилища некоторое подмножество данных, отбираемых по указанным значениям одного или нескольких измерений. Например, для сравнения результатов за текущий и предыдущий месяцы требуется информация только о двух месяцах из 36 (в нашей учебной базе данных). Специализированные схемы индексации разрабатываются таким образом, чтобы обеспечить сверхбыструю выборку соответствующих строк из таблицы фактов и объединение их со строками таблиц измерений. В некоторых из этих схем используются технологии побитовой обработки, когда за каждым из возможных значений измерения (или комбинации измерений) закрепляется отдельный разряд в значении индекса. Строки, соответствующие условию отбора, можно очень быстро идентифицировать с помощью операций побитовой арифметики, поскольку побитовое сравнение выполняется быстрее, чем сравнение чисел.