Министерство науки и высшего образования Российской Федерации
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

О.Е. АВРУНЕВ, В.М. СТАСЫШИН

МОДЕЛИ БАЗ ДАННЫХ

Утверждено Редакционно-издательским советом университета
в качестве учебного пособия

НОВОСИБИРСК
2018

УДК 004.652(075.8)
А 219

Рецензенты:

Ю.В. Тракимус, канд. техн. наук, доцент каф. ПМт,
А.А. Цыгулин, канд. техн. наук, доцент каф. ТПИ

Работа подготовлена на кафедре теоретической
и прикладной информатики для бакалавров IV курса ФПМИ
дневного отделения (направления 01.03.02, 02.03.03) и магистров
I курса ФПМИ дневного отделения (направление 02.04.03).

Аврунев О.Е.

А 219 Модели баз данных: учебное пособие / О.Е. Аврунев, В.М. Стасышин. – Новосибирск: Изд-во НГТУ, 2018. – 124 с.

ISBN 978-5-7782-3749-0

Предлагаемое учебное пособие содержит описание широкого спектра моделей баз данных: дореляционных, реляционных, постреляционных, в том числе моделей класса NoSQL. Несмотря на то что реляционная модель в настоящее время является доминирующей, каждая из этих моделей занимает свою нишу в области информационных технологий при решении задач обработки данных.

Включенный в пособие материал входит в программы курсов лекций «Технологии баз данных» (02.03.03), «Базы данных и экспертные системы» (01.03.02), «Современные технологии баз данных» (02.04.03), читаемых студентам факультета прикладной математики и информатики.

Учебное пособие может быть полезно также специалистам, занимающимся информационными технологиями и самостоятельно осваивающим вопросы разработки баз данных.

УДК 004.652(075.8)

ISBN 978-5-7782-3749-0

ОГЛАВЛЕНИЕ

В основе любой базы данных лежит модель данных. ***Моделью данных*** называется формализованное описание структур единиц информации и операций над ними в информационной системе. Тип модели данных определяет логическую структуру базы данных и то, каким образом данные могут быть сохранены, организованы и обработаны.

1. Ранние подходы к организации баз данных

К ранним моделям относят модели, предшествующие реляционной модели данных: иерархическую, сетевую модели и модель на основе инвертированных списков. В явном виде таковых моделей осталось очень мало. Однако эти системы исторически предшествовали реляционным, и для правильного понимания причин повсеместного перехода к реляционным моделям необходимо иметь о них представление. Ниже приведены наиболее общие характеристики ранних систем.

1. Эти системы активно использовались в течение многих лет, дольше, чем используются многие из реляционных СУБД. Некоторые из ранних систем используются даже в наше время, накоплены громадные базы данных, и одной из актуальных проблем информационных систем является использование этих систем совместно с современными системами.

2. Все ранние системы не основывались на каких-либо абстрактных моделях. Понятие модели данных фактически вошло в обиход специалистов в области баз данных только вместе с реляционным подходом. Абстрактные представления ранних систем появились позже на основе анализа и выявления общих признаков у различных конкретных систем.

3. В ранних системах доступ к базам данных производился на уровне записей. Пользователи этих систем осуществляли явную нави-

гацию в базе данных, используя языки программирования, расширенные функциями СУБД. Интерактивный доступ к базе данных поддерживался только путем создания соответствующих прикладных программ с собственным интерфейсом.

4. Навигационная природа ранних систем и доступ к данным на уровне записей заставляли пользователя самого производить всю оптимизацию доступа к базе данных без какой-либо поддержки системы.

5. После появления реляционных систем большинство ранних систем было оснащено «реляционными» интерфейсами. Однако в большинстве случаев это не сделало их по-настоящему реляционными системами, поскольку оставалась возможность манипулировать данными в естественном для них режиме.

Необходимость изучения ранних моделей вызвана еще одним обстоятельством: внутренняя организация реляционных систем во многом основана на использовании методов ранних систем, и некоторое знание в области ранних систем будет полезно для понимания путей развития постреляционных СУБД (например, в основе графовой модели лежит сетевая модель данных).

1.1. Иерархическая модель данных

Иерархическая модель данных была исторически первой структурой баз данных, видимо, из-за того, что древовидные иерархические структуры широко используются в повседневной человеческой деятельности. Это всевозможные классификаторы, ускоряющие поиск информации, иерархические функциональные структуры управления и т.д.

Иерархическая модель данных – представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней.

В 1968 г. компания IBM предложила систему управления информацией IMS (Information Management System), основанную на иерархической модели данных. В этой модели данные представляются как дерево связанных записей. Имеется один корневой (родительский) тип записи – корень дерева. С ним в подчиненной связи типа 1: N находятся дочерние записи. Связи между записями выражаются в виде отношений предок/потомок, а у каждой записи есть ровно одна родительская запись. Это помогает поддерживать ссылочную целостность:

когда запись удаляется из дерева, все ее потомки должны быть также удалены. Общая схема иерархической модели представлена на рис. 1.1.

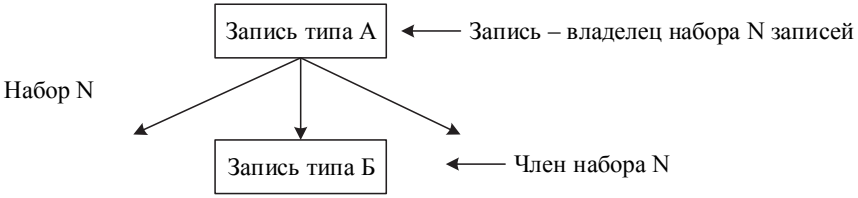


Рис. 1.1. Общая схема иерархической модели данных

Одной из наиболее важных сфер применения первых СУБД было планирование производства в компаниях, занимающихся выпуском продукции. Например, если автомобильная компания хотела выпустить 10 000 автомобилей одной модели и 5000 автомобилей другой модели, необходимо знать, сколько деталей следует заказать у своих поставщиков. Чтобы ответить на этот вопрос, необходимо выяснить, из каких частей состоит изделие, затем определить, из каких деталей состоят эти части, и т.д. Так, например, автомобиль состоит из двигателя, корпуса и ходовой части, двигатель состоит из клапанов, цилиндров, свеч и т.д. Список составных частей изделия (рис. 1.2) по своей природе является иерархической структурой. Для хранения данных, имеющих такую структуру, и была разработана иерархическая модель данных, представляющая собой упорядоченный граф (дерева).

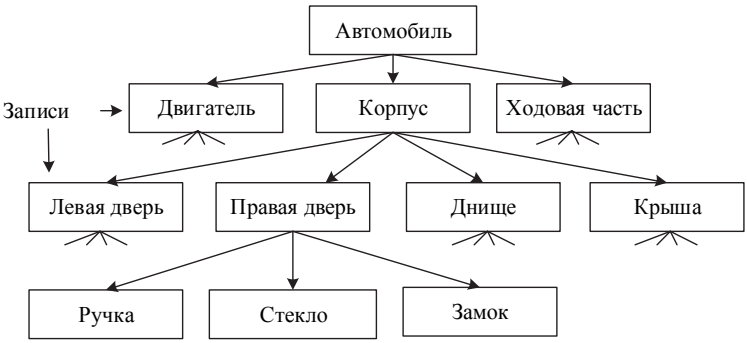


Рис. 1.2. Список составных частей изделия

В этой модели каждая *запись* базы данных представляла конкретную деталь. Между записями существовали *отношения предок/потомок,* связывающие каждую часть с деталями, входящими в нее.

Другими примерами такой иерархии могут быть представление структуры учебной дисциплины (рис. 1.3), где имеются уровни иерархической структуры: дисциплина, раздел, тема, вопрос, или структура организации (директор, заместитель, руководители отделов, сотрудники) (рис.1.4).

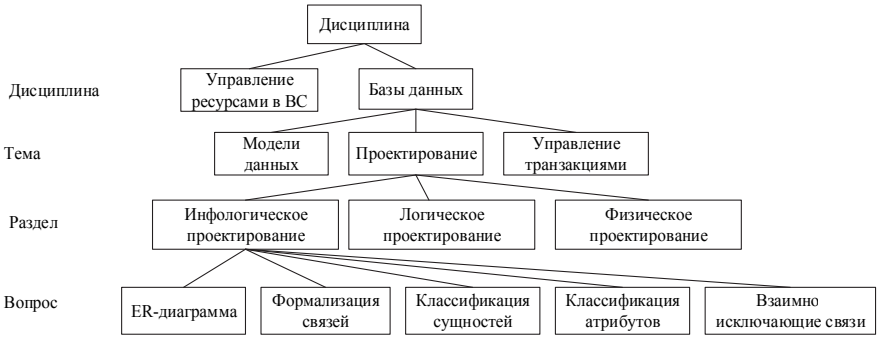


Рис. 1.3. Структура учебной дисциплины

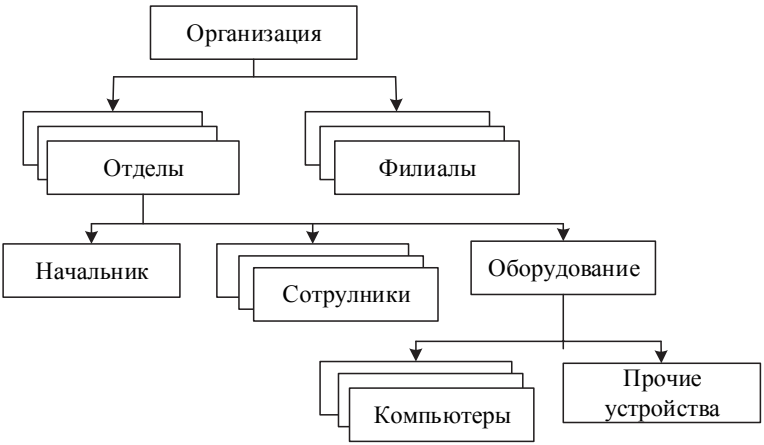


Рис. 1.4. Структура организации

На рис. 1.5 изображена простая иерархическая база данных, в которой фиксируется деятельность исполнителей. Корень дерева представляет собой запись об исполнителе. Ее потомками являются две записи о счет-фактурах и три записи об оплатах счетов. Структура счета номер 362 уточняется в трех дочерних записях, у счета номер 368 – в одной записи.

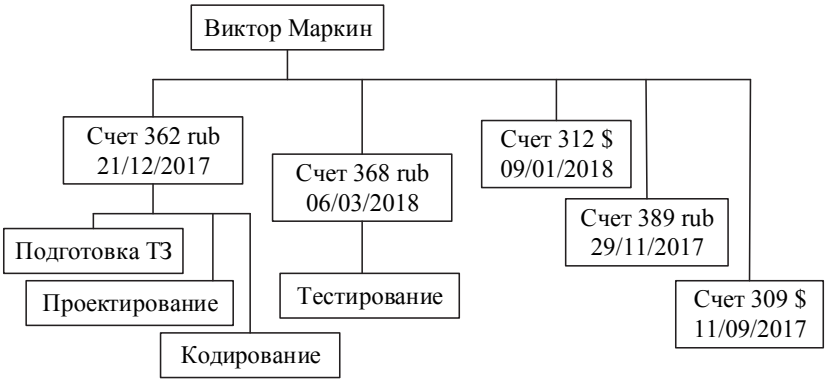


Рис. 1.5. Экземпляры логических записей базы данных организации

В графической диаграмме схемы базы данных вершины используются для интерпретации сущностей, а дуги – для интерпретации связей. При этом возможна ситуация, когда сущность-предок не имеет потомков или имеет их несколько, тогда как у сущности-потомка обязательно должен быть только один предок. Объекты, имеющие общего предка, называются *близнецами*.

Для описания структуры (или, правильнее, схемы) иерархической базы данных на некотором языке программирования используется тип данных *дерево*. Тип данных *дерево* схож с типами *структура* языков программирования PL/1 или C и *запись* языка Паскаль.

Тип *дерево* является составным. Он может включать в себя подтипы, каждый из которых, в свою очередь является типом *дерево*. Каждый из типов *дерево* состоит из одного *корневого* типа и упорядоченного (возможно, пустого) набора подчиненных типов. Каждый из элементарных типов, включенных в тип *дерево*, является простым или составным типом *запись*. Простая запись состоит из одного типа,

например, числового, а составная *запись* объединяет некоторую совокупность типов (целое, строку символов и т.д.).

На рис. 1.6 показан пример дерева, отражающего схему иерархической базы данных. Здесь тип записи «Отдел» является предком для типов записей «Руководитель» и «Сотрудник», а «Руководитель» и «Сотрудник» – потомки типа записи «Отдел». Между типами поддерживаются связи.

Рис. 1.6. Схема иерархической базы данных организации

Содержимое базы данных с такой схемой могло бы выглядеть, например так, как показано на рис. 1.7.

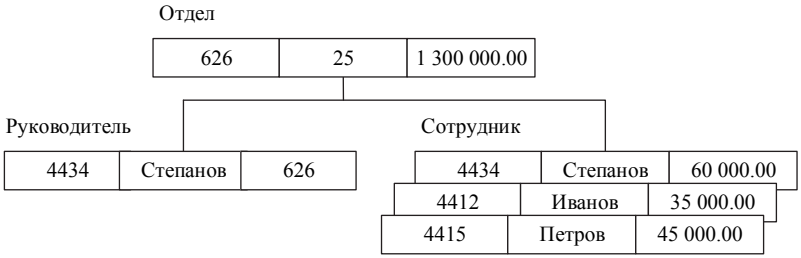


Рис. 1.7. Экземпляры логических записей иерархической базы данных организации

Еще один пример схемы базы данных с иерархической моделью, описывающей структуру больницы, и частично загруженной базы данных с такой схемой показаны на рис. 1.8 и 1.9.

В иерархических СУБД может использоваться терминология, отличная от приведенной выше. Так в упомянутой системе IMS понятию *запись* соответствует термин *сегмент*, а под *записью базы данных* понимается вся совокупность записей, относящихся к одному экземпляру типа *дерево*.

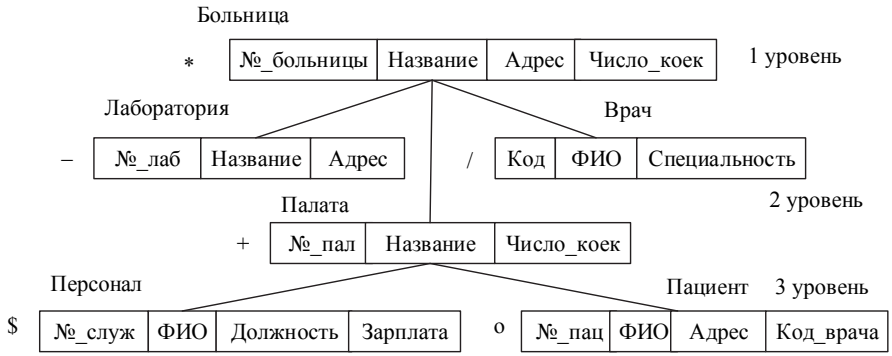


Рис. 1.8. Схема иерархической базы данных больницы

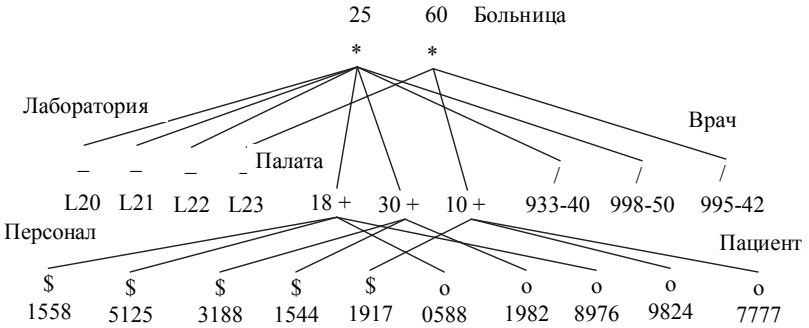


Рис. 1.9. Экземпляры логических записей иерархической базы данных больницы

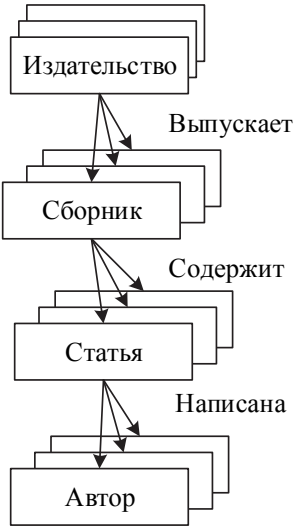
Иерархические базы данных имеют централизованную структуру, поэтому безопасность данных легко контролировать. К сожалению, определенные знания о физическом порядке хранения записей все же необходимы, так как отношения предок/потомок реализуются в виде физических указателей из одной записи на другую. Это означает, что поиск записи осуществляется методом прямого обхода дерева. Записи, расположенные в одной половине дерева, ищутся быстрее, чем в другой. Отсюда следует необходимость правильно упорядочивать записи, чтобы время их поиска было минимальным.

Основные операции манипулирования иерархически организованными данными:

- найти указанное дерево базы данных (например, дерево со значением 40 в поле «№_больницы»);
- перейти от одного дерева к другому;
- перейти от одной записи к другой внутри дерева в порядке иерархии (например, от палаты к первому пациенту);
- перейти от одной записи к другой на одном уровне внутри дерева (например, к следующему пациенту);
- вставить новую запись в указанную позицию;
- удалить текущую запись.

Способы описания схем данных, базирующиеся на иерархической модели, и соответствующие языки манипулирования данными зависят от конкретной реализации.

На рис. 1.10 приведено графическое представление иерархической модели базы данных тематических сборников издательств и описание схемы иерархической модели в системе IMS фирмы IBM.



Dbd Name = Тематические сборники
Segm Name = Издательство
Field Name = Название
Field Name = Адрес
Field Name = Счет
Segm Name = Сборник, Parent = Издательство
Field Name = Название
Field Name = Периодичность
Field Name = Цена
Field Name = Ответственный_редактор
Segm Name = Статья, Parent = Сборник
Field Name = Название
Segm Name = Автор, Parent = Статья
Field Name = ФИО
Field Name = Гонорар

Рис. 1.10. Описание иерархической базы данных тематических сборников издательств

Язык доступа к данным, который поддерживает IMS, позволяет обращаться к элементам напрямую, зная название и, возможно, дополнительное условие. Например, можно распечатать название всех сборников, ответственным редактором которых является Иванов:

```
Get Unique Сборник Where Ответственный_редактор = «Иванов»
/* получили первый сборник */
While true do
    Print Сборник.Название
    Get next Сборник Where Ответственный_редактор = «Иванов»
End while
```

Выбрав один из сборников в предыдущем примере, можно спуститься «вниз» по иерархии (оператор **Get next Within Parent** позволяет перебрать все элементы-потомки, соответствующие выбранному элементу данного уровня) и просмотреть одну или несколько статей из выбранного сборника.

```
Get Unique Сборник Where Ответственный_редактор = «Иванов»
/* получили первый сборник */
While true do
    Print «Сборник», Сборник.Название
    Get next Within Parent Статья
    While true do
        Print Статья.Название
        Get next Within Parent Статья
    End while
    Get next Сборник Where Ответственный_редактор = «Иванов»
End while
```

Достоинства иерархической модели

1. *Простота модели*. Принцип построения баз данных в иерархической модели легок для понимания. Иерархия базы данных напоминает структуру компании или генеалогическое дерево.

2. *Использование отношений предок/потомок*. Иерархическая модель позволяет легко представлять отношения предок/потомок, например, «А является частью В» или «А принадлежит В».

3. *Быстродействие*. В иерархической модели отношения предок/потомок реализуются в виде физических указателей из одной записи на другую, поэтому перемещение по базе данных происходит достаточно быстро. Поскольку структура данных в иерархической модели отличается простотой, СУБД может размещать записи предков и потомков на диске рядом друг с другом, что позволяло свести к минимуму количество операций чтения-записи.

Недостатки иерархической модели

1. Операции манипулирования данными в иерархических системах ориентированы прежде всего на поиск информации сверху-вниз, т.е. по данному экземпляру сегмента-отца можно найти все экземпляры сегментов-сыновей. Обратный поиск затруднен, а часто и невозможен. Например, попытка реализовать запрос типа «В скольких сборниках статей опубликовал свои статьи господин Петров?» может оказаться весьма трудной задачей.

2. Дублирование данных на логическом уровне.

3. Для представления связи M:N необходимо дублирование деревьев (рис. 1.11).

4. В иерархической модели автоматически поддерживается целостность ссылок между предками и потомками по правилу: никакой потомок не может существовать без своего родителя. Целостность по ссылкам между записями, не входящими в одну иерархию, не поддерживается. Поэтому невозможно хранение в базе данных порожденного узла без соответствующего исходного. Аналогично, удаление исходного узла влечет удаление всех порожденных узлов (деревьев), связанных с ним.

Помимо упомянутой СУБД IMS фирмы IBM к иерархическим СУБД относятся также Time-Shared Date Management System (компа-

ния Development Corporation), Mark IV MultiAccess Retrieval System (компания Control Data Corporation), System 2000 разработки SAS-Institute, из отечественных в 80-е годы прошлого столетия были широко популярны СУБД Ока, ИНЭС, МИРИС.

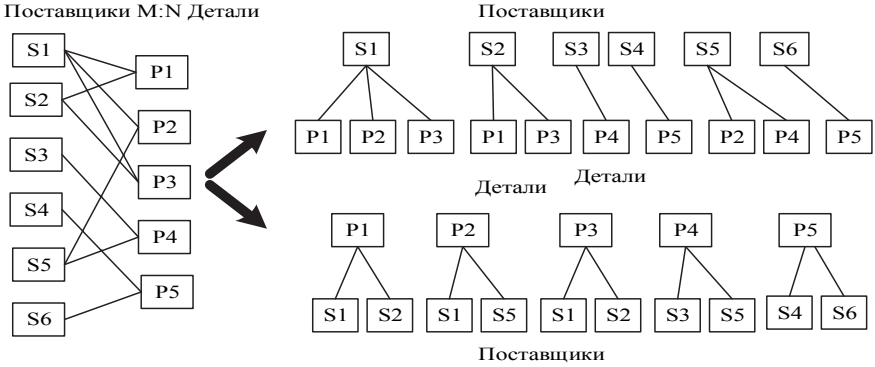


Рис. 1.11. Дублирование деревьев для представления связи M:N

Другими примерами использования иерархической модели являются:

- иерархические файловые системы, состоящие из корневого каталога и иерархии подкаталогов и файлов; иерархической базой данных является *Каталог папок Windows*, с которым можно работать, запустив *Проводник ОС Windows*;
- *реестр Windows*, в котором хранится вся информация, необходимая для нормального функционирования компьютерной системы (данные о конфигурации компьютера и установленных драйверах, сведения об установленных программах, настройки графического интерфейса и др.);
- серверы каталогов, такие как LDAP и Active Directory.

Еще одним примером иерархической базы данных является база данных *Доменная система имен компьютеров*, подключенных к Интернету (рис. 1.12).

На верхнем уровне находится табличная база данных, содержащая перечень доменов верхнего уровня, из которых 7 – административные, а остальные 257 – географические. На втором уровне находятся табличные базы данных, содержащие перечень доменов второго уровня

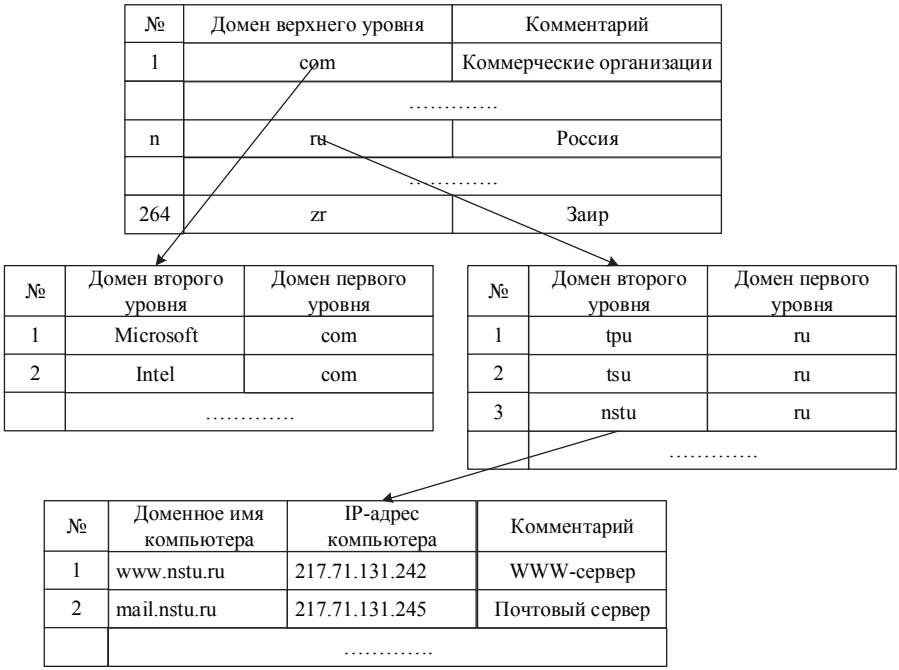


Рис. 1.12. Иерархическая база данных «Доменная система имен»

для каждого домена первого уровня. На третьем уровне могут находиться табличные базы данных, содержащие перечень доменов третьего уровня для каждого домена второго уровня, и таблицы, содержащие IP-адреса компьютеров, находящихся в домене второго уровня.

Контрольные вопросы и упражнения

1. Каковы структурные элементы иерархической модели данных?
2. Как обеспечивается связь в иерархической модели данных?
3. Как реализуется отношение M:N в иерархической модели данных?
4. Каковы достоинства и недостатки иерархической модели данных?
5. Чем обеспечивается высокая эффективность доступа к данным в иерархической модели данных?

6. Перечислите операции манипулирования иерархически организованными данными.

7. Опишите схему иерархической модели, приведенную на рис. 1.4, на языке СУБД IMS (необходимые поля записей задайте произвольно).

8. Опишите схему иерархической модели, приведенную на рис. 1.8, на языке СУБД IMS (необходимые поля записей задайте произвольно).

9. Как обеспечиваются ограничения целостности в иерархической СУБД?

10. Приведите примеры использования иерархических моделей данных.

11. Смоделируйте структуру реляционной базы данных, хранящую схему иерархической модели, приведенную на рис. 1.4.

12. Средствами реляционной базы данных смоделируйте структуру, хранящую схему иерархической модели, приведенную на рис. 1.8.

13. Что понимается под термином «сегмент» в иерархической модели данных СУБД IMS?

1.2. Сетевая модель данных

Если структура данных оказывалась сложнее, чем простая иерархия, простота организации иерархической базы данных становилась ее недостатком.

Например, в базе данных для хранения заказов один заказ может участвовать в трех различных отношениях предок/потомок, связывающих заказ с клиентом, разместившим его, со служащим, принявшим его, и с заказанным товаром. Такая структура (рис. 1.13) не соответствует иерархической модели.

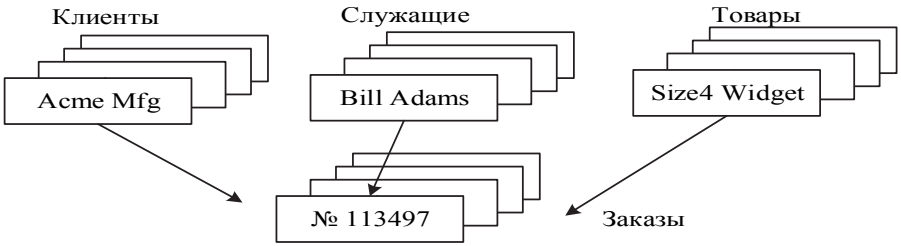


Рис. 1.13. Неиерархическая структура базы данных

Сетевой подход к организации данных является расширением иерархического. Цель разработчиков сетевой модели – создание модели, позволяющей описывать связи M:N, чтобы одна запись могла участвовать в нескольких отношениях предок/потомок (рис. 1.14).

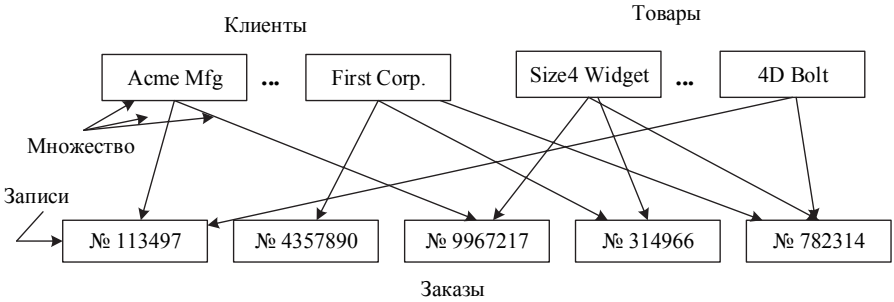


Рис. 1.14. Участие записи в нескольких отношениях предок/потомок

При различных способах реализации сетевых моделей наибольшее распространение получила модель CODASYL (Conference on Data Systems Languages), предложенная группой Data Base Task Group (DBTG) Комитета по языкам программирования, 1971 г.

СУБД, построенные на сетевой модели: DMS (компания UNIVAC), DBMS (компания DEC), IDMS (компания Culliname). Из современных сетевых СУБД можно отметить dbVista под DOS и Windows.

Сетевая модель данных базируется также на использовании представления данных в виде графа. С точки зрения теории графов сетевой модели соответствует произвольный граф: если в иерархической модели запись-потомок должна иметь в точности одного предка, то в сетевой модели данных потомок может иметь любое число предков. Вершины графа используются для интерпретации типов объектов, дуги графа используются для интерпретации типов связей между типами объектов.

Структурными элементами сетевой модели данных CODASYL являются элемент данных, агрегат данных, запись, набор записей (рис. 1.15).

Элемент данных – наименьшая поименованная единица данных (аналог поля в файловых системах). Значение элемента данных может

быть числовым (разных типов), символьным, логическим, может быть неопределенным, имя используется для идентификации (табельный номер, шифр детали, год рождения и т.д.).

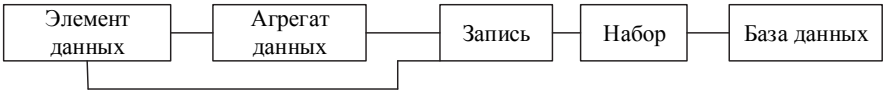


Рис. 1.15. Структурные элементы сетевой модели данных CODASYL

Агрегат данных – поименованная совокупность элементов данных внутри записи, которую можно рассматривать как единую целую. Имя агрегата используется для его идентификации в схеме структуры более высокого уровня. Агрегат может быть простым, если состоит только из элементов данных, и составным, если включает в себя другие агрегаты (рис. 1.16).

Дата		
Число	Месяц	Год

Предприятие			
Название	Адрес		
	Почтовый индекс	Город	Улица Номер дома

Рис. 1.16. Примеры агрегатов данных

Различают агрегаты типа *вектор* и типа *повторяющаяся группа*. Если в агрегате повторяющаяся компонента является простым элементом данных, его называют вектором. Пример вектора – агрегат «Заработная плата», в которой экземпляр элемента данных может повторяться до 12 раз (по числу месяцев в году). Агрегат, повторяющаяся компонента которого представлена совокупностью данных, называется повторяющейся группой. В повторяющуюся группу могут входить отдельные элементы данных, векторы, агрегаты или повторяющиеся группы. На рис. 17 представлен агрегат «Заказ на покупку», имеющий в своем составе повторяющуюся группу «Партия товара».

Запись – поименованная совокупность элементов данных и/или агрегатов, которая не входит в состав никакого другого агрегата (рис. 1.18). Запись может иметь сложную иерархическую структуру, допускает многократное применение агрегации. Имя записи использу-

ется для идентификации типа записи в структурах более высокого уровня.

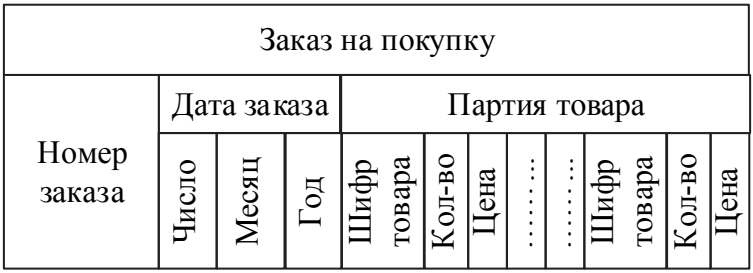


Рис. 1.17. Пример повторяющейся группы

Рис. 1.18. Пример агрегата *Дата* в записи *Сотрудник*

Связи между записями выполняют так называемые наборы. ***Набор*** – поименованная двухуровневая иерархическая структура, связывающая запись-владельца и записи-членов. Каждый набор должен содержать только один экземпляр записи-владельца и любое количество экземпляров записей-членов (рис. 1.19).

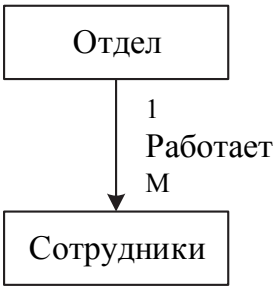


Рис. 1.19. Пример набора, связывающего две записи

Структура сетевой базы данных основана на следующих правилах.

1. База данных содержит любое количество типов записей и типов наборов.

2. Между двумя типами записей может быть определено любое количество типов наборов.

3. Тип записи может быть владельцем и одновременно членом нескольких типов наборов

На рис. 1.20 представлены три типа записей: «Отдел», «Служащие» и «Руководитель» и три типа связей (три набора): «Состоит из служащих», «Имеет руководителя» и «Является служащим».

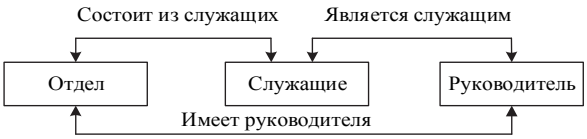


Рис. 1.20. Записи и связи в сетевой модели

Пример схемы сетевой базы данных приведен на рис. 1.21.

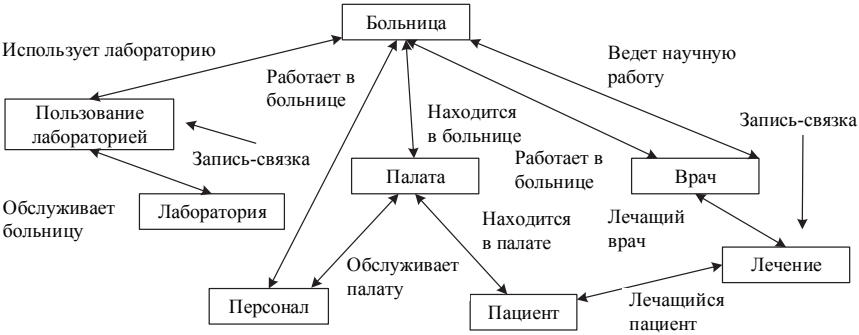


Рис. 1.21. Схема сетевой базы данных

Наборы могут быть нескольких разновидностей.

- С одними и теми же типами записей, но разными типами наборов. На рис. 1.21 присутствуют различные наборы записей между одними и теми же типами записей «Врач» и «Больница» (может быть еще и финансовый аспект).

• Наборы из трех или более записей, в том числе и с обратной связью. На рис. 1.21 это записи «Врач», «Пациент» и «Лечение».

• Так называемые сингулярные наборы (рис. 1.22), у которых нет естественного владельца и в качестве его выступает система.

Рис. 1.22. Сингулярный набор

В дальнейшем такие наборы могут приобрести запись-владельца.

• Наборы между несколькими типами записей, называемые многочленными наборами, которые представляют собой отношение между тремя или более типами записей, один из которых назначается владельцем, а остальные – членами набора. На рис. 1.23 приведен пример многочленного набора, владельцем которого является запись типа «Лектор», а членами – записи типа «Методическая работа», «Научные труды», «Тезисы докладов».

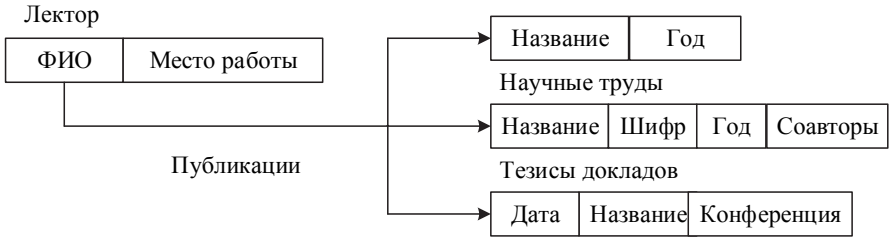


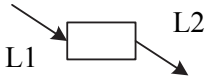
Рис. 1.23. Многочленный набор

Ограничения на типы записей и связей сетевой модели

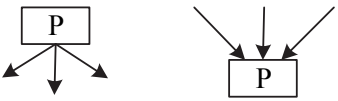
1. Все типы связей должны быть функциональными (1:1, 1:M, M:1).

2. Экземпляр записи может быть членом только одного экземпляра набора среди всех экземпляров набора одного типа (он может входить в состав двух и более экземпляров наборов, но разных типов).

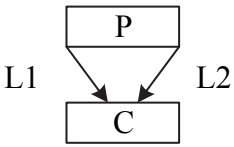
3. Экземпляр записи может быть потомком в одном наборе L1 и предком в другом наборе L2.



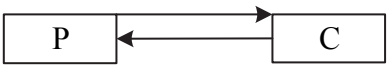
4. Экземпляр записи P может быть предком в любом числе наборов, и аналогично, может быть потомком в любом числе наборов.



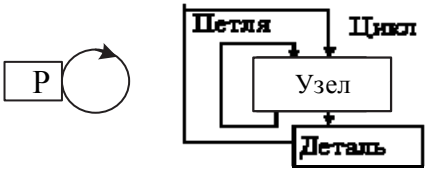
5. Может существовать любое число типов наборов с одним и тем же типом записи предка и одним и тем же типом записи потомка. Например, если L1 и L2 – два типа набора с одним и тем же типом записи предка P и одним и тем же типом записи потомка C, то правила, по которым образуется родство, в разных связях могут различаться.



6. Записи P и C могут быть предком и потомком в одном наборе и потомком и предком – в другом.



7. Предок и потомок могут быть одного типа записи. Сетевая модель может содержать циклы, когда предшествующая вершина является в то же время предыдущей. Связь записей одного типа называется петлей.



8. Для представления связи M:N вводится дополнительный тип записи и две функциональные связи типа 1:M и M:1. При необходимости запись-связка может содержать дополнительную информацию (рис. 1.24).

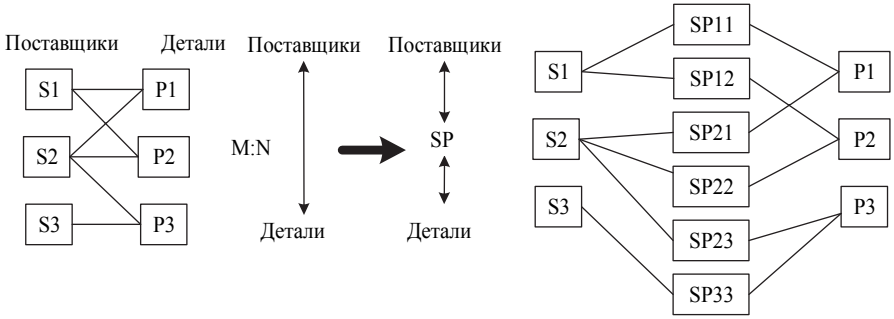


Рис. 1.24. Представление связи M:N

Операции манипулирования данными:

- найти запись в наборе однотипных записей (палату с указанным номером);
- перейти от предка к первому потомку по некоторой связи (к первому врачу некоторой больницы);
- перейти к следующему потомку в некоторой связи (от врача Сидорова к Иванову);
- перейти от потомка к предку по некоторой связи (найти больницу врача Сидорова);
- создать новую запись, уничтожить запись, модифицировать запись;
- включить в связь, исключить из связи;
- переставить в другую связь и т.д.

Если вернуться к примеру с изданием тематических сборников и попытаться расширить его, чтобы он более полно соответствовал реальным взаимоотношениям, то схема сетевой модели будет выглядеть так, как показано на рис. 1.25.

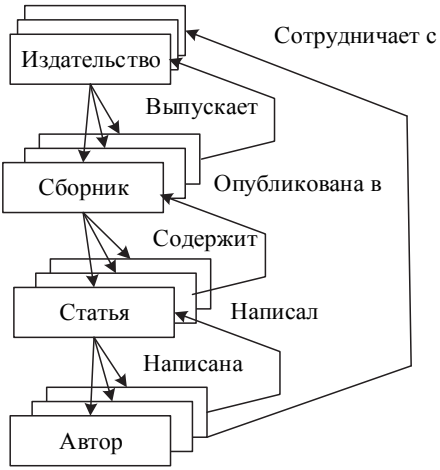


Рис. 1.25. Схема сетевой базы данных тематических сборников издательств

Описание приведенной сетевой схемы на языке CODASYL в этом случае примет вид:

Record name is Издательство
01 Название Type is Character 30
01 Адрес Type is Character 30
01 Счет Type is Picture "999999999"
Record name is Сборник
01 Название Type is Character 30
01 Периодичность Type is fixed
01 Цена Type is fixed
01 Ответственный_редактор Type is Character 20
Record name is Статья
01 Название Type is Character 80

Record name is Автор
01 ФИО Type is Character 20
01 Гонорар Type is fixed
Set name is Выпускает
Owner is Издательство
Member is Сборник
Set name is Содержит
Owner is Сборник
Member is Статья
. . . . . .

Любая сетевая структура может быть приведена к более простому виду путем введения избыточности (рис. 1.26). В некоторых случаях возникающая при этом избыточность мала и является допустимой, в других случаях она может быть чрезмерной.

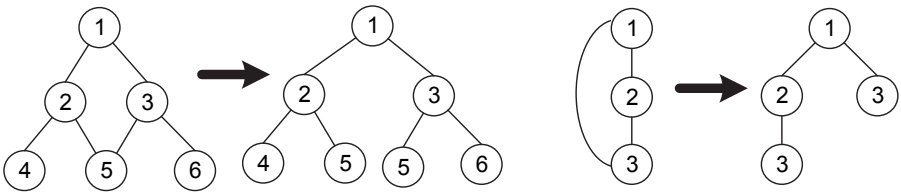


Рис. 1.26. Преобразование сетевой модели путем введения избыточности

Контрольные вопросы и упражнения

1. Каковы структурные элементы сетевой модели данных?
2. Что такое элемент данных? Агрегат? Запись?
3. Перечислите основные виды агрегатов.
4. Как обеспечивается связь между записями в сетевой модели данных?
5. Каковы правила построения сетевой базы данных?
6. Перечислите ограничения на типы записей и связей сетевой модели.
7. Приведите пример многочленного набора данных, отличающегося от приведенного на рис. 1.23.

8. Приведите пример, в котором между двумя типами записей может быть определено несколько типов наборов.

9. Приведите пример, в котором записи являются предком и потомком в одном наборе, и потомком и предком – в другом.

10. Каковы достоинства и недостатки сетевой модели данных?

11. Перечислите операции манипулирования данными сетевой модели.

12. Опишите схему сетевой модели базы данных, приведенной на рис. 1.21, на языке CODASYL (необходимые поля записей задайте произвольно).

13. Как задаются ограничения целостности в сетевой СУБД?

14. Средствами реляционной базы данных смоделируйте структуру, хранящую схему сетевой модели, приведенной на рис. 1.21.

1.3. Системы, основанные на инвертированных списках

Сложность практического использования иерархических и сетевых СУБД заставляла искать иные способы представления данных. В конце 1960-х годов появились СУБД на основе инвертированных файлов, отличающиеся простотой организации и наличием весьма удобных языков манипулирования данными.

Известными представителями таких систем являются Datacom/DB (компания Applied Data Research, Inc.) и Adabas (компания Software AG). Они применяются, как правило, на больших компьютерах фирмы IBM.

Организация доступа к данным на основе инвертированных списков используется практически во всех современных реляционных СУБД, но в реляционных СУБД пользователи не имеют непосредственного доступа к инвертированным спискам (индексам).

База данных на инвертированных списках похожа на реляционную базу данных, т. е. также состоит из таблиц отношений, однако ей присущи важные отличия:

- допускается сложная структура атрибутов (атрибуты не обязательно атомарны);
- строки таблиц (записи) упорядочены в некоторой последовательности, каждой строке присваивается уникальный номер; физиче-

ская упорядоченность строк всех таблиц может определяться и для всей базы данных (так делается, например, в СУБД Datacom/DB);

- пользователям видны и хранимые таблицы, и пути доступа к ним;
- пользователь может управлять логическим порядком строк в каждой таблице с помощью специального инструмента – индексов; эти индексы автоматически поддерживаются системой и явно видны пользователям.

Некоторые атрибуты могут быть объявлены поисковыми (ключевыми), для каждого из них создается индекс, который содержит упорядоченные значения ключей и указатели на соответствующие записи основной таблицы (инвертированный список). Если таблицу требуется упорядочить по нескольким ключам, то создается несколько индексов. Если, например, в основной таблице с данными, представленной на рис. 1.27, ключевыми атрибутами являются атрибуты «Фамилия» и «Группа», то инвертированные списки выглядят так, как показано на рис. 1.28.

Адреса записей	**Имена столбцов (поля)**		
	Номер зачетной книжки	**Фамилия**	**Группа**
100	052693109	Гриценко	ПМИ-61
220	052693020	Медведев	ПМИ-62
250	052387415	Нотова	ПМ-41
300	050102608	Матвеев	ПМ-41
400	052931417	Мастихин	ПМ-71
440	050102212	Медведев	ПМ-22
530	052693216	Матвеев	ПМИ-62
600	052931712	Паничев	ПМИМ-71
700	052931107	Кочанов	ПМИ-71

Рис. 1.27. Основная таблица с данными

Ключевой атрибут	Индекс	Адреса записей (инвертированные списки)
Фамилия	Гриценко	100
	Кочанов	700
	Мастихин	400
	Матвеев	300, 530
	Медведев	220, 440
	Нотова	250
	Паничев	600
Группа	ПМИМ-71	600
	ПМИ-61	100
	ПМИ-62	220, 530
	ПМИ-71	700
	ПМ-22	440
	ПМ-41	250, 300
	ПМ-71	400

Рис. 1.28. Индексы и инвертированные списки

Тогда для поиска записей по условию Фамилия = «Медведев» и Группа = «ПМИ-62» достаточно найти пересечение списков с соответствующими индексами: адрес искомой записи = (220, 440) ^ (220, 530) = 220.

Системой поддерживаются два класса операций над данными:

- поиск адреса записи по некоторому пути доступа и некоторому условию;
- обновление, удаление или выборка записи с заданным адресом.

Типичный набор операторов:

- LOCATE FIRST – найти первую запись таблицы T в физическом порядке; возвращается адрес записи;
- LOCATE FIRST WITH SEARCH KEY EQUAL – найти первую запись таблицы T с заданным значением ключа поиска K; возвращается адрес записи;
- LOCATE NEXT – найти первую запись, следующую за записью с заданным адресом в заданном пути доступа; возвращается адрес записи;
- LOCATE NEXT WITH SEARCH KEY EQUAL – найти следующую запись таблицы T в порядке пути поиска с заданным значением K; при этом должно быть установлено соответствие между используемым способом сканирования и ключом K; возвращается адрес записи;

- LOCATE FIRST WITH SEARCH KEY GREATER – найти первую запись таблицы T в порядке ключа поиска K со значением ключевого поля, бóльшим заданного значения K; возвращается адрес записи;
- RETRIVE – выбрать запись с указанным адресом;
- UPDATE – обновить запись с указанным адресом;
- DELETE – удалить запись с указанным адресом;
- STORE – включить запись в указанную таблицу; операция генерирует адрес записи.

К достоинствам рассмотренного метода построения базы данных следует отнести:

- более быстрый поиск (особенно поиск уникальной записи по нескольким условиям);
- возможность хранения элементов данных со сложной структурой.

Недостаток модели – отсутствие строгого математического аппарата, отсутствие средств для описания ограничений целостности базы данных и, как следствие – большая трудоемкость программирования запросов к базе данных. В некоторых системах поддерживаются ограничения уникальности значений некоторых полей, но в основном все возлагается на прикладную программу. Кроме этого, такие СУБД обладают рядом ограничений на количество файлов для хранения данных, количество связей между ними, длину записи и количество ее полей.

Однако СУБД Adabas хороша в задачах, в которых надо из очень большого объема данных выбрать по сложному запросу относительно небольшое количество данных (например, найти автомобиль, иномарка, то ли серого, то ли голубого цвета, седан, номер московский, цифровая его часть оканчивается на 78, водитель среднего возраста, в очках).

Контрольные вопросы и упражнения

1. Что представляет собой инвертированный список?
2. Как строится инвертированный список?
3. Перечислите набор операторов систем на основе инвертированных списков.
4. Каковы достоинства и недостатки систем на основе инвертированных списков?
5. Каким образом инвертированные списки используются в реляционных СУБД?

2. Реляционная модель данных

Реляционная модель предложена Э. Коддом (E. Kodd, математик, сотрудник IBM, 1970 г., статья «A Relational Model of Data for Large Shared Data Banks») на основе теории отношений (relation) и опирается на систему понятий, важнейшими из которых являются тип данных, домен, атрибут, кортеж, первичный и внешний ключ. Принципы, используемые в реляционной модели, вытекают из понятия n-арного отношения, представляющего собой подмножество декартового произведения.

Факторы, обеспечившие быстрое распространение реляционной модели:

- с прагматической точки зрения база данных представляется в виде двумерных таблиц (отношений), обработка в которых не зависит от организации хранения данных в памяти;
- с математической точки зрения реляционная база данных – конечный набор отношений различной арности, являющихся областью приложений математической логики, теории множеств и общей алгебры; замкнутость реляционной модели (операции над отношениями дают отношение) обеспечивает основу для интерпретации выводимости, избыточности и непротиворечивости данных;
- наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения.

2.1. Основные понятия реляционной модели данных

Основными понятия реляционных баз данных являются тип данных, домен, атрибут, кортеж, отношение, первичный ключ (рис. 2.1).

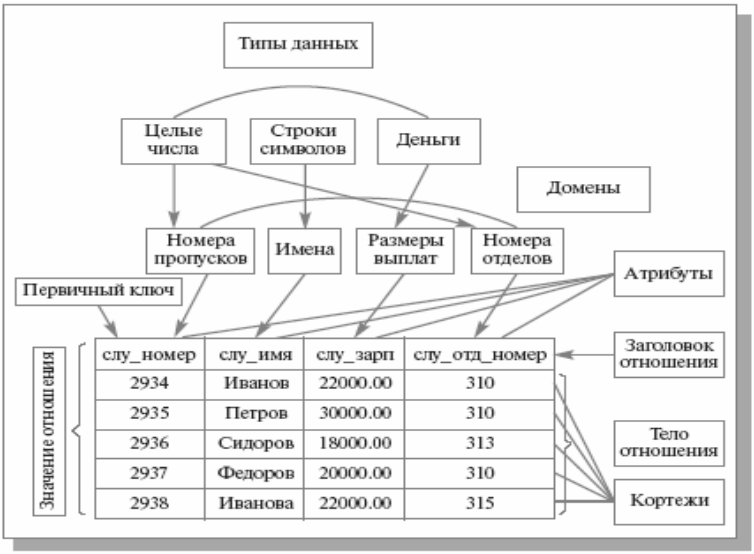


Рис. 2.1. Основные понятия реляционной модели данных

Понятие ***тип данных*** в реляционной модели данных полностью адекватно понятию типа данных в языках программирования. Обычно в современных реляционных базах данных допускается хранение символьных, числовых данных, битовых строк, специализированных числовых данных (таких как «деньги»), а также специальных «темпоральных» (temporal) данных (дата, время, временной интервал). В примере на рис. 2.1 представлены данные трех типов: строки символов, целые числа и «деньги».

Домен – допустимое потенциальное множество значений данного типа. В самом общем виде домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат «истина», то элемент данных является элементом домена. Например, домен «Номера пропусков» в приведенном примере определен на базовом типе целых чисел, но в число его значений могут входить только те целые числа, которые являются номерами существующих отделов. Семантическая нагрузка понятия домена состоит в том, что данные

считаются сравнимыми только в том случае, когда они относятся к одному домену. В приведенном примере значения доменов «Номера пропусков» и «Номера отделов» относятся к типу целых чисел, но не являются сравнимыми.

Каждый домен описывает некоторый атрибут. ***Атрибут*** – некоторая характеристика объекта (сущности). Атрибуты имеют имена, через которые к ним производится обращение. Имя атрибута должно быть уникальным внутри отношения.

Таким образом, имеется совокупность множеств D_1, D_2,. . . D_n, не обязательно различных, называемых доменами n-арного отношения. Совокупность значений $<d_1, d_2, \ldots d_n,>$, где $d_i \in D_i$, i = 1, n называется ***кортежем.*** Множество всех кортежей $<d_1, d_2, \ldots d_n,>$, где $d_i \in D_i$, i = 1, n арности n составляет декартово произведение множеств $D_1 \times D_2 \times \ldots \times D_n$. Тогда ***отношение R*** на множествах D_1, D_2,. . . D_n – это некоторое подмножество кортежей арности n декартового произведения доменов $D_1 \times D_2 \times \ldots \times D_n$.

Схема отношения **(*заголовок отношения*)** – именованное конечное множество упорядоченных пар вида <A, T>, где A является именем атрибута, а T обозначает имя некоторого базового типа или ранее определенного домена. Иногда схему отношения обозначают множеством атрибутов, предполагая, что атрибут несет информацию о домене.

Сотрудники ({Слу_номер, номера пропусков}, {Сотр_имя, имена}, {Сотр_зарпл, размер з/п}, {Отдел_ном, номера отделов})

Число атрибутов отношения носит название **степени** или «арности» схемы отношения (степень отношения **Сотрудники** равна 4).

Мощность тела отношения – число кортежей отношения.

Реляционная база данных – это конечное множество отношений, записываемое в виде:

$\mathbf{R_1(A_{11}, A_{12} \ldots A_{1n})}$ где A_{ij} пара {имя атрибута, имя домена}

. . . . . .

$\mathbf{R_m(A_{m1}, A_{m2} \ldots A_{mz})}$

Обычным житейским представлением отношения является таблица, заголовком которой является схема отношения, а строками – кортежи отношения; в этом случае имена атрибутов именуют столбцы этой таблицы.

С математической точки зрения отношение является множеством и не может содержать совпадающих элементов, а следовательно, в любой момент времени никакие два кортежа отношения не могут быть дубликатами друг друга. Чтобы это обеспечить, в отношении должен присутствовать некоторый атрибут (или набор атрибутов), однозначно определяющий каждый кортеж отношения и обеспечивающий уникальность кортежей. Атрибут, значение которого идентифицирует кортеж, называется ***ключом*** отношения (в отношении **Сотрудники** это атрибут **Слу_номер**). Если отношение имеет несколько ключей, один из них объявляется ***первичным ключом*** (primary key).

Свойства первичного ключа:

- *уникальность*: в любой момент времени никакие два кортежа отношения не должны иметь одно и то же значение;
- *минимальность*: ни один из атрибутов не может быть исключен из набора атрибутов первичного ключа без нарушения свойств уникальности.

В зависимости от количества атрибутов, входящих в ключ, различают простые и сложные (составные) ключи.

Простой ключ – ключ, содержащий только один атрибут. Как правило, в качестве него используют самый короткий и простой из возможных типов данных (целочисленный тип), при этом операции, использующие ключ (операции объединения), выполняются значительно быстрее.

Сложный (составной) ключ – ключ, состоящий из нескольких атрибутов.

Суперключ – сложный ключ, с большим числом атрибутов, не удовлетворяющий свойству минимальности. Используется крайне редко, когда избыточность может оказаться полезной пользователю.

С точки зрения информативности атрибута (атрибутов), составляющего первичный ключ, различают искусственные и естественные ключи.

Искусственный или суррогатный ключ – ключ создаваемый самой СУБД или пользователем с помощью некоторой процедуры, который сам по себе не содержит информации. Используется для создания уникальности идентификаторов строк. Им также заменяют слишком сложные ключи. Как правило, пользователю они не показываются.

Естественный ключ – ключ, содержащий только значимые атрибуты, содержащие информацию.

К достоинствам естественных ключей можно отнести то, что они несут вполне определенную информацию, и их использование не приводит к необходимости добавлять к таблице атрибуты, значения которых для пользователя не несут никакого смысла и используются только для связи между отношениями, что позволяет получить более компактную форму таблиц.

Основным же недостатком естественных ключей является то, что их использование весьма затруднительно в случае изменения предметной области. Значения атрибутов первичного ключа не должно изменяться, и однажды заданное значение первичного ключа для кортежа не может быть изменено. Это требование необходимо для поддержания ссылочной целостности базы данных, которая устанавливается по первичному ключу.

Другим недостатком естественных ключей является то, что, как правило, они являются составными и содержат строковые атрибуты, что сказывается на скорости выполнения операций над данными.

В любой из таблиц может оказаться несколько наборов атрибутов, которые можно выбрать в качестве ключа, такие наборы называются ***потенциальными*** и ***возможными*** ключами.

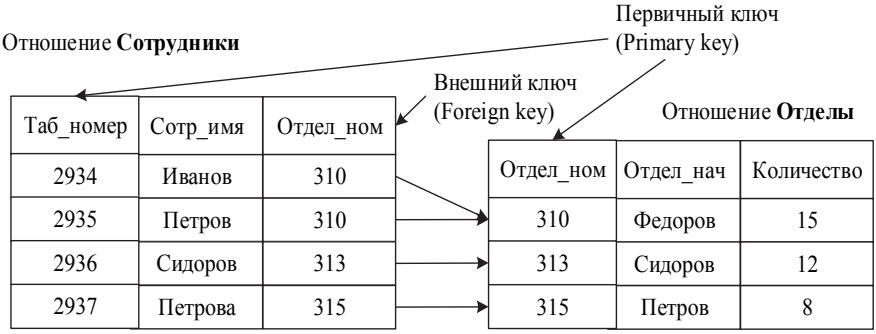
Вторичные ключи – ключи, отличные от комбинации атрибутов первичного ключа. Они могут не обладать свойством уникальности. Наконец, ***перекрывающиеся ключи*** – сложные ключи, которые имеют один или несколько общих столбцов.

Выбор ключа – это не формальный момент. От того, что выбрано в качестве ключа, зависят задаваемые ограничения целостности модели. Если, например, в отношении

Сотрудники (Слу_номер, Сотр_имя, Сотр_зарпл, Отдел_ном)

первичным ключом определен атрибут **– Слу_номер**, это означает, что в организации не должно быть сотрудников с одинаковыми табельными номерами. Если в этом же отношении первичным ключом определен составной ключ **{Слу_номер, Отдел_ном}**, это означает, что в разных отделах могут работать служащие с одинаковыми номерами, но в каждом отделе номера служащих различны.

Для организации взаимосвязи отношений используется понятие внешнего ключа (foreign key). Атрибут называется ***внешним ключом***, если его значения однозначно характеризуют сущности, представленные кортежами некоторого другого отношения, заданные своим первичным ключом (рис. 2.2).



Таб_номер	Сотр_имя	Отдел_ном
2934	Иванов	310
2935	Петров	310
2936	Сидоров	313
2937	Петрова	315

Отдел_ном	Отдел_нач	Количество
310	Федоров	15
313	Сидоров	12
315	Петров	8

Рис. 2.2. Связь отношений через внешний ключ

При проектировании базы данных следует различать понятия базового и производного отношения. *Базовое отношение* – это отношение, которое содержит один или несколько атрибутов, характеризующих свойства объекта, а также первичный ключ, *производное отношение* – это отношение, которое не характеризует свойства объекта и используется для обеспечения связей между другими таблицами.

2.2. Фундаментальные свойства отношений

1. *Отсутствие кортежей-дубликатов.*

Данное свойство следует из определения отношения как множества кортежей (в классической теории множеств по определению каждое множество состоит из различных элементов). Отсюда же вытекает наличие у каждого отношения первичного ключа.

Во многих практических реализациях реляционных СУБД допускается нарушение свойства уникальности кортежей для промежуточных отношений, порождаемых неявно при выполнении запросов. Такие отношения являются не множествами, а мультимножествами.

2. *Отсутствие упорядоченности кортежей*.

Это свойство также является следствием определения отношения как множества кортежей.

3. *Отсутствие упорядоченности атрибутов*.

Атрибуты отношений не упорядочены, поскольку по определению схема отношения есть множество пар {имя атрибута, имя домена}.

4. *Атомарность значений атрибутов*.

Базовым требованием для классических реляционных баз данных является требование нормализованных отношений или отношений, представленных в первой нормальной форме. А отношения, представленные в первой нормальной форме, предполагают атомарность (неделимость) атрибутов. Исходный вариант отношения **Сотрудники** на рис. 2.1 является нормализованным отношением. Потенциальный пример ненормализованного отношения приведен на рис. 2.3. Значением атрибута **Отдел** в нем является отношение.

НОМЕР_ОТДЕЛА	ОТДЕЛ		
	СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП
310	2934	Иванов	22000.00
	2935	Петров	30000.00
	2937	Федоров	20000.00
313	2936	Сидоров	18000.00
315	2938	Иванова	22000.00

Рис. 2.3. Пример ненормализованного отношения

Хотя нормализованные отношения обладают некоторыми ограничениями (не любую информацию удобно представлять в виде плоских таблиц), они существенно упрощают манипулирование данными.

Рассмотрим два идентичных оператора занесения кортежа.

- Зачислить сотрудника Кузнецова (Таб. номер 3000, зарплата 115,000) в отдел номер 320.
- Зачислить сотрудника Кузнецова (Таб. номер 3001, зарплата 115,000) в отдел номер 310.

Если информация о сотрудниках представлена в виде нормализованного отношения **Сотрудники**, оба оператора будут выполняться одинаково (вставить кортеж в отношение **Сотрудники**). Если же работать с ненормализованным отношением **Отделы**, то первый оператор выразится в занесение кортежа, а второй – в добавление информации о Кузнецове в множественное значение атрибута **Отдел** кортежа с первичным ключом 310.

Наиболее распространенная трактовка реляционной модели данных принадлежит К. Дейту. Согласно Дейту реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода:

- структурная часть,
- манипуляционная часть
- целостная часть.

В структурной части модели фиксируется, что единственной структурой данных, используемой в реляционных базах данных, является нормализованное n-арное отношение.

В манипуляционной части модели утверждаются два фундаментальных механизма манипулирования реляционными базами данных – реляционная алгебра и реляционное исчисление. Первый механизм базируется, в основном, на классической теории множеств и является процедурным языком: задает алгоритм (перечень операций) получения результата, а второй механизм базируется на классическом логическом аппарате исчисления предикатов первого порядка и является непроцедурным языком, указывая свойства результата, но не указывая алгоритма его достижения.

В целостной части реляционной модели данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД.

1. Первое требование называется требованием **целостности сущностей** (*entity integrity*). Любой сущности в реляционной базе данных соответствует кортеж отношения. И требование состоит в том, чтобы любой кортеж любого отношения был отличим от любого другого кортежа этого отношения. Это означает, что любое отношение должно обладать первичным ключом.

Данное требование автоматически удовлетворяется, если не нарушаются базовые свойства отношений.

Более строго требование целостности сущностей полностью звучит следующим образом: в любом отношении должен существовать первичный ключ, и никакое значение первичного ключа в кортежах отношения не должно содержать неопределенных значений.

На практике не все характеристики сущности могут быть известны к тому моменту, когда требуется зафиксировать сущность в базе данных. Простой пример – процедура принятия на работу человека, размер заработной платы которого еще не определен. В этом случае сотрудник отдела кадров, который заносит в отношение **СЛУЖАЩИЕ** кортеж, описывающий нового служащего, просто не может ввести значение атрибута **СЛУ_ЗАРП**, поскольку любое значение будет неверным.

Э. Кодд предложил использовать в таких случаях неопределенные значения. ***Неопределенные значения*** **(*Null-значения*)** не принадлежат ни к какому типу данных и могут присутствовать среди значений любого атрибута, определенного на любом типе данных, если только это явно не запрещено при определении атрибута.

Поскольку требование целостностей сущности означает, что первичный ключ должен полностью идентифицировать каждую сущность, в первичном ключе не допускаются неопределенные значения.

В классической реляционной модели это требование распространяется и на *возможные (уникальные) ключи (Unique)*. В SQL-ориентированных СУБД это требование для возможных ключей не поддерживается. Возможный ключ может содержать Null-значения, однако трактовка Null-значений уникальных ключей в различных СУБД может существенно различаться.

2. Второе требование называется ***требованием целостности по ссылкам*** (*referential integrity*).

Требование целостности по ссылкам, или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть неопределенным. Для нашего примера из рис. 2.2 это означает, что если для сотрудника указан номер отдела, то этот отдел должен существовать.

Как ограничения целостности сущностей и ограничения по ссылкам поддерживаются СУБД?

Для соблюдения целостности по сущностям достаточно гарантировать отсутствие в любом отношении кортежей с одним и тем же значением первичного ключа.

Для соблюдения целостности по ссылкам:

- при обновлении ссылающегося отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа;
- при удалении кортежа из отношения для обеспечения целостности по ссылкам на практике существуют три подхода:

– запрещается производить удаление кортежа, на который имеются ссылки; сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа;

– при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным;

– третий подход (каскадное удаление) состоит в том, что при удалении кортежа из основного отношения с некоторым значением внешнего ключа автоматически удаляются кортежи из связанного с ним отношения с таким же значением первичного ключа.

Э. Коду принадлежат 12 правил, которым должна удовлетворять реляционная база данных.

1. *Правило информации.* Вся информация на логическом уровне представляется только значениями в таблицах без использования указателей и индексов.

2. *Правило гарантированного доступа*. Каждый элемент таблицы должен быть доступен через комбинацию из имени таблицы, имени поля и ключа.

3. *Системная поддержка неопределенных значений*. Для представления отсутствующих данных с любым типом используются Null-значения.

4. *Динамический каталог, создаваемый на основе реляционной модели*. Метаданные (словари) формируются тем же языком, что и данные. Пользователи, обладающие соответствующими правами, могут

работать с метаданными с помощью того же реляционного языка, который они применяют для работы с основными данными.

5. *Правило исчерпывающего подъязыка данных*. В базе данных возможно использование нескольких языков взаимодействия с базой данных (например, язык вопросов-ответов), однако один, чаще всего SQL, должен быть главным и поддерживать все следующие элементы:

– определение данных,

– определение представлений,

– обработку данных (интерактивную и программную),

– задание условий целостности данных,

– идентификацию прав доступа,

– задание границ транзакций (начало, завершение и отмена).

6. *Правило обновления представления* (*view*). Все обновляемые представления должна обновлять и система.

7. *Ввод, обновление и удаление данных*. Возможность работать с отношением как с одним операндом должна существовать не только при чтении данных, но и при добавлении, обновлении и удалении данных.

8. *Физическая независимость данных*. Возможное изменение расположения базы данных, способов хранения данных не должно оказывать влияния на прикладные программы и пользователя.

9. *Логическая независимость данных*. При добавлении или удалении элементов (таблиц, полей) в структуре базы данных другие части базы данных остаются неизменными.

10. *Независимость условий целостности*. Должна существовать возможность определять условия целостности в языке реляционной базы данных и хранить их в каталоге, а не в прикладной программе.

11. *Независимость распределения*. В распределенных базах данных расположение данных независимо. Для пользователя такая база данных должна выступать как локальная база данных.

12. *Правило соблюдения правил*. Нельзя обходить ограничения, введенные с помощью языка SQL.

Контрольные вопросы и упражнения

1. Что такое тип данных? Домен? Атрибут? Кортеж? Схема отношения? Отношение?

2. Каков порядок строк отношения?

3. Какими способами задается домен?

4. Чем отличается домен от типа данных?

5. Что такое схема базы данных?

6. Для чего используются первичные ключи? Как они задаются?

7. В чем разница между первичным и возможным ключом?

8. Какими свойствами должен обладать первичный ключ?

9. Дайте классификацию ключей с точки зрения информативности входящих в них атрибутов.

10. Что такое суррогатный ключ?

11. В чем различие вторичных и первичных ключей?

12. Какова арность отношения P (таблица деталей), используемого при выполнении лабораторных работ?

13. Какова мощность тела отношения SPJ (таблица поставок), используемого при выполнении лабораторных работ?

14. Что такое базовое отношение? Что такое производное отношение?

15. Что такое внешний ключ? Какими свойствами он должен обладать?

16. Какой тип связи получится, если в качестве внешнего ключа будет выступать первичный ключ?

17. Какой тип связи установится при использовании неуникального внешнего ключа?

18. Перечислите фундаментальные свойства отношений.

19. Почему важна атомарность атрибутов?

20. Что понимается под структурной, манипуляционной, целостной частями реляционной модели данных?

21. Какой смысл имеет Null-значение?

22. Каково значение выражения «IsNull(0)»?

23. Каков результат выполнения операции умножения над двумя полями, содержащими значения Null?

24. Как обеспечивается целостность по сущностям?

25. Как обеспечивается целостность по ссылкам?

26. Для таблиц, используемых при выполнении лабораторных работ, определите атрибуты, кортежи, домены и ключи отношений.

27. Для таблиц, используемых при выполнении лабораторных работ, определите первичный ключ и выполните его классификацию.

28. Перечислите правила Э. Кода и дайте им интерпретацию.

3. Объектно-реляционная модель данных

Объектно-реляционная модель данных реализована с помощью реляционных таблиц, но включает объекты, аналогичного понятию объекта в объектно-ориентированном программировании. Возникновение объектно-реляционной модели данных объясняется тем, что реляционные базы данных хорошо работают со встроенными типами данных и гораздо хуже с пользовательскими, нестандартными. Перестройка СУБД с целью включения в нее поддержки нового типа данных – не лучший выход из положения. Вместо этого объектно-реляционная СУБД позволяет загружать код, предназначенный для обработки «нетипичных» данных. Таким образом, база данных сохраняет свою табличную структуру, но способ обработки некоторых полей таблиц определяется извне, т.е. программистом.

В объектно-реляционной модели данных используются такие объектно-ориентированные компоненты, как пользовательские типы данных, инкапсуляция, полиморфизм, наследование, переопределение методов и т.п.

Категория объектно-реляционных СУБД позволяет:

- поддерживать сложные типы данных;
- вводить новые типы данных;
- наследовать объекты: типы данных, таблицы и пр.

Объектно-реляционные свойства имеют все широко известные СУБД, в том числе Oracle Database, Informix, DB2, PostgreSQL.

К сожалению, до настоящего времени разработчики не пришли к единому мнению о том, что должна обеспечивать объектно-реляционная модель данных. В 1999 г. был принят стандарт SQL-99, а в 2003 г. вышел второй релиз этого стандарта, получивший название SQL-3, который определяет основные характеристики объектно-реляционной

модели данных. Но до сих пор объектно-реляционные модели, поддерживаемые различными производителями СУБД, существенно отличаются друг от друга.

Учитывая сказанное, описание объектно-реляционной модели будет излагаться в нотации СУБД Informix.

3.1. Сложные типы данных

Сложные типы данных подразделяются на три категории:

- коллекции;
- строчные типы;
- типы данных, определенные пользователем.

В свою очередь коллекции делятся на множества (SET), мультимножества (MULTISET) и списки (LIST), а строчные типы делятся на именованные и неименованные.

Коллекция представляет собой сложный тип, составленный из отдельных элементов, каждый из которых принадлежит к одному и тому же типу данных, при этом элементы, в свою очередь, могут быть сложными типами, встроенными типами или типами, определенными пользователем.

Коллекция SET является множеством уникальных неупорядоченных элементов (без дублирования). Коллекция MULTISET не упорядочена, подобно SET, но допускает дублирующие значения:

```
create table flags    (item_number int,
                       country varchar(40),
                       colors multiset (char(10) not null));
```

Вставка строк выглядит следующим образом:

```
insert into flags values (1234, "Russia", multiset{"While", "Blue", "Red"});
insert into flags values (3456, "USSR", multiset{"Red"});
```

При выборке строк можно искать значение конкретного элемента множества. Запрос

```
select item_number, country from flags where "Red" in (colors);
```

вернет две строки

Items_number	*country*
1234	*Russia*
3456	*USSR,*

а запрос

select item_number, country from flags where “Blue” in (colors);

вернет одну строку

Items_number	*country*
1234	*Russia*

В отличие от SET и MULTISET коллекция LIST является упорядоченным множеством элементов, где допустимы дублирующие значения.

Ниже приведен пример создания таблицы для хранения средних температур для каждого месяца года и занесения в нее данных:

```
create table city_temp      (city_name char(30),
                             avg_temp list (integer not null));
insert into city_temp values (“Novosibirsk”, List{-22.5, -18.7, -14.5, . . . .});
```

В отличие от коллекций строчные типы являются группами элементов разных типов и формируют шаблон записи, который впоследствии можно использовать при определении таблиц:

```
create row type address_t   (street char(30),
                             city char(20),
                             state char(2),
                             zip char(5));
create table customer       (customer_no integer,
                             name char(30),
                             address address_t,
                             balance money);
```

Для вставки данных в таблицу, включающую строчный тип, используется оператор «::»:

insert into customer values (12345, “John Smith”,

row(“355 27h Street”, “San Diego”, “CA”, “12355”)::address_t, 1000.00);

insert into customer values (22222, “Fred Brown”,

row(“355 Oak Street”, “Fairoaks”, “CA”, “92555”)::address_t, 2000.00);

Для манипуляции отдельными полями строчного типа используется точечная нотация. Запрос

select customer_no, name from customer where address.state=”CA”;

вернет результат

Customer_no	*name*
12345	*John Smith*
22222	*Fred Brown*

При обновлении строчного типа в запросе необходимо указывать все поля, а не только те, которые обновляются. Этим подчеркивается, что данные поля не меняются:

update customer set

address=row(address.street, address.city, “AZ”, address.zip)::ad-dress_t

where customer_no=22222;

Строчные типы можно использовать внутри других строчных типов:

create row type zip_t (code char(5),
suffix char(4));

create row type address_t (street char(30),
city char(20),
state char(2),
zip zip_t);

```
create table customer      (customer_no integer,
                            name char(30),
                            address address_t,
                            balance money);
```

В этом случае для вставки необходимо выполнить приведение двух строчных типов

```
insert into customer values (12345, “John Smith”,
                row(“355 27h Street”, “San Diego”, “CA”,
                row(“94131”, “2011”)::zip_t)::address_t, 1000.00);
```

Все ранее использованные в примерах – именованные типы. Ниже приведен пример неименованного типа:

```
create table customer      (customer_no integer,
                            name char(30),
                            address row      (street char(30),
                            city char(20),
                            state char(2),
                            zip char(5));
                            balance money);
```

3.2. Наследование

Наследование дает возможность одному объекту приобретать свойства другого объекта, например, если одна таблица определяется как подтаблица другой, то она наследует поведение супертаблицы (ограничения, триггеры, индексы, функции и пр.). Аналогично строчный тип, определенный как подтип базового типа, наследует поля данных и поведение супертипа. Ниже приведен пример введения строчного типа для служащих с базовой информацией, строчного типа для служащих разных категорий: продавцов, работающих на основе комиссионных и управляющих различными счетами, инженеров, получающих премию и обладающих определенными навыками. При этом

введенные строчные типы **salesman** и **engineer** наследуют поля **name, dept, salary**.

```
create row type employee_t          (name char(20),
                                     dept int,
                                     salary money);
create row type salesman_t (quota money,
                            attainment decimal(4,2),
                            commission decimal(4,2),
                            accounts set (varchar(30) not null))
                    under employee_t;
create row type engineer_t (bonus money,
                            skills set (varchar(30) not null))
                    under employee_t;
```

Определение таблиц на основе созданных типов будет выглядеть следующим образом:

```
create table employee of type employee_t;

create table salesman of type salesman_t under employee_t;

create table engineer of type engineer_t under employee_t;
```

Подтаблицы **salesman** и **engineer** наследуют все свойства супертаблицы **employee**, включая столбцы, индексы и любые ограничения на таблицу.

При вставке строки в одну из подтаблиц необходимо вставлять и данные для столбцов, унаследованных из супертаблицы:

```
insert into employee values (“Fred Smith”, 111, 1000.00);

insert into salesman values (“Tom Jones”, 222, 2000.00,
    50000.00, 200.00, 3.00, set{“ABC Shoes”, “Leiner Boots”});

insert into engineer values (“Janet Brown”, 333, 3000.00,
    1000.00, set{“Cobol”, “Java”});
```

В запросе на выборку необходимо учитывать иерархии таблиц.

Оператор select для супертаблицы будет возвращать данные для всех столбцов, определенных в супертаблице и для унаследованных столбцов в подтаблице:

select name, dept from employee;

Результат запроса

Name	*dept*
Fred Smith	*111*
Tom Jones	*222*
Janet Brown	*333*

Если запрос нужно ограничить только супертаблицей, запрос

select name, dept from only(employee);

возвратит результат

Name	*dept*
Fred Smith	*111*

Для строчных типов, определенных в иерархии, могут быть разработаны подпрограммы, определенные пользователем и учитывающие принадлежность к тем или иным типам.

Пример. Разным категориям служащих зарплата начисляется различными способами: административные служащие получают базовую зарплату, продавцы получают базовую зарплату плюс комиссионные в процентах с продаж, инженеры получают базовую зарплату плюс премию. Базовая функция будет иметь вид:

```
create function compensation (emp employee_t) returning money;
    return emp.salary;
end function;
```

Если сейчас применить эту функцию, то она будет использоваться одинаковым образом для всех типов, возвращая базовую зарплату для всех категорий служащих:

select name, salary, compensation (e) compensation from employee e;

Name	*Salary*	*Compensation*
Fred Smith	*$1000.00*	*$1000.00*
Tom Jones	*$2000.00*	*$2000.00*
Janet Brown	*$3000.00*	*$3000.00*

Теперь определим функции для разных типов, определенных в условии примера:

create function compensation (sales salesman_t) returning money;

*return (sales.salary+sales.commission*quota/100+sales.attainment);*

end function;

create function compensation (eng engineer_t) returning money;

return (eng.salary+eng.bonus);

end function;

Применение функции теперь даст разные результаты для разных типов:

select name, salary, compensation (e) compensation from employee e;

Name	*Salary*	*Compensation*
Fred Smith	*$1000.00*	*$1000.00*
Tom Jones	*$2000.00*	*$3700.00*
Janet Brown	*$3000.00*	*$4000.00*

Когда для типов определяется подпрограмма, то менеджер функций ядра СУБД ищет подпрограмму с сигнатурой передаваемой функции. Если подпрограммы с такой сигнатурой нет, то используется подпрограмма, унаследованная от супертипа.

3.3. Определенные пользователем типы данных и функция приведения

Пример. Для хранения значений температур в США и ЮАР используются определенные пользователем типы, базирующиеся на целом типе данных, соответственно **fahrenheit** и **celsius**, и две таблицы со средними зимними и летними температурами

create distinct type fahrenheit as integer;

create distinct type celsius as integer;

create table rsa_city (city_name char(20),
population int,
avg_summer_temp celsius,
avg_winter_temp celsius);

create table usa_city (city_name char(20),
population int,
avg_summer_temp fahrenheit,
avg_winter_temp fahrenheit);

Поля **Avg_summer** и **Avg_winter** и в одной, и в другой таблице – целые числа, но суть их различна. Как сравнивать эти типы? **Функция приведения (casting)** выполняет операции, необходимые для преобразования данных одного типа в другой. Приведения могут быть:

- неявные; они выполняются сервером с использованием множества встроенных функций приведения между всеми базовыми типами;
- явные; между новым типом и типом-источником.

В случае явных функций приведения необходимо использовать конструкцию «<источник>::<новый тип>» или функцию «Cast (источник as новый тип)».

insert into rsa_city values ("Capetown", 3200000, 21::celsius, 14::celsius);

insert into rsa_city values ("Johannesburg", 1900000, 19::celsius, 9::celsius);

insert into usa_city values ("Anchorage", 250000, 45::fahrenheit, 20:: fahrenheit);

insert into usa_city values ("Miami", 2100000, 82:: fahrenheit, 67:: fahrenheit);

Но этого достаточно для приведения в определенный пользователем тип (celsius) из базового типа, на основе которого он определен, поскольку описание **create type** определяет функцию приведения. Для сравнения данных таблиц **rsa_city** и **usa_city** можно рекомендовать два подхода.

Первый способ состоит в том, чтобы

- создать функцию приведения для выполнения преобразований между типами данных **celsius** и **fahrenheit**:

create function c_to_f (temp celsius) returning fahrenheit;

*return (9*temp::integer/5+32):: fahrenheit;*

end function;

- зарегистрировать данную функцию для сервера как функцию приведения:

create explicit cast (celsius as fahrenheit with c_to_f);

Теперь сервер знает, как сравнивать типы **celsius** и **fahrenheit**, и можно сравнивать температуры в разных городах. Оператор SQL

select city_name, avg_summer_temp:: fahrenheit avg_summer

from rsa_city order by 2 desc

union

select city_name, avg_summer_temp avg_summer

from usa_city order by 2 desc;

породит результат

City_name	*Avg_summer*
Miami	*82*
Capetown	*69*
Johannensburg	*66*
Anchorage	*45*

При этом показываемые температуры – значения по Фаренгейту.

Второй путь сравнения **celsius** и **fahrenheit** основан на создании перегружаемых функций. Как пример, можно создать функцию, которая бы вычисляла, была ли температура выше или ниже точки замерзания. Поскольку вычисления разнятся в зависимости от того, была ли температура зарегистрирована по Цельсию или по Фаренгейту, можно использовать перегружаемую функцию:

```
create function above_freezing (temp celsius ) returning boolean;
if (temp::integer >0)        then return 't' :: boolean;
                             else 'f' :: boolean;  Endif
end  function;
create function above_freezing (temp fahrenheit) returning boolean;
if (temp::integer >32)       then return 't' :: boolean;
                             else 'f' :: boolean;
endif
end  function;
```

Ниже показан пример использования данной функции для поиска городов с теплым климатом (зимняя t °C > 0°):

```
select city_name,  avg_winter_temp  from usa_city
            where above_freezing (avg_winter_temp) order by 2 desc
union
select city_name,  avg_winter_temp:: Fahrenheit from rsa_city
            where above_freezing (avg_winter_temp) order by 2 desc;
```

Данный запрос покажет зимнюю температуру по Фаренгейту, переведенную из температуры по Цельсию:

City_name	*Avg_winter_temp*
Miami	*67*
Capetown	*57*
Johannesburg	*48*

Контрольные вопросы и упражнения

1. Что такое сложный тип данных «Set»? Как его задать?
2. Что такое сложный тип данных «Multiset»? Как его задать?
3. Что такое сложный тип данных «List»? Как его задать?
4. Как выполняется наследование типов? В каких случаях это может быть полезно?
5. Как выполняется наследование таблиц?
6. Как пользователь может ввести собственные типы данных?
7. Что такое явные функции приведения? Как они задаются?
8. Как определить неявную функцию приведения?
9. Что такое перегружаемая функция?
10. Как можно использовать перегружаемые функции для сравнения типов данных?

4. Объектно-реляционное отображение

4.1. Традиционный метод доступа

Для многих клиентских приложений учетных систем, реализованных в двухзвенной архитектуре, основными элементами логики и интерфейса являются списки в табличном виде и карточки документа объекта. Каждому из этих элементов может быть сопоставлен запрос, являющийся источником данных, и запросы, выполняющиеся при модификации данных.

В таком случае приведенные выше проблемы не оказывают большого влияния на эффективность разработки. Но как только возникает необходимость усложнения функциональности приложения, как правило, влекущая и усложнение архитектуры, и появление дополнительных слоев, непосредственное формирование и выполнение SQL запросов приводит к увеличению объема программного кода и трудностям в его сопровождении из-за следующих недостатков:

- смешивание логики получения информации из базы данных и логики обработки полученной информации;
- у строки SQL-запроса отсутствует семантика в контексте приложения;
- проблема рассогласования схемы данных базы данных и структуры классов приложения.

Основным архитектурным шаблоном является «Модель, Представление, Контроллер (Model View Controller)». «Модель» в этом случае представляет собой набор классов, моделирующих предметную область приложения. Как следствие, для этих классов необходимо реализовать наиболее эффективное сохранение и получение информации из базы данных.

4.2. Основные компоненты объектно-реляционного отображения

При использовании объектно-реляционного отображения (кратко, ORM) возможны два способа создания модели классов, при первом, более традиционном подходе, называемом также «Database First» классы определяются для существующей структуры таблиц. В случае подхода «Model First» для созданной в приложении модели классов автоматически генерируются таблицы базы данных.

В приложении должно быть задано соответствие между классами и таблицами, и между каждым атрибутом класса и полем соответствующей таблицы. Это отображение в зависимости от конкретной библиотеки, реализующей объектно-реляционное отображение, может быть задано либо в отдельном конфигурационном файле, либо с помощью декораторов (если язык программирования поддерживает такую возможность), или специальных атрибутов класса. Последний способ реализован в ORM-библиотеке SQLAlchemy для языка Python.

Рассмотрим применение объектно-реляционного отображения в приложении на примере модели данных о сотрудниках, работающих в различных организациях. Данные о физическом лице хранятся в таблице person, данные об организациях – в таблице corporation, а связь многие ко многим между ними, представляющая факт работы человека в организации, – в таблице **staff**.

Структура классов, соответствующая данной модели, приведена ниже.

```
class Person(Base):
    _tablename_ = 'person'
    id = Column(Integer, primary_key=True)
    family_name = Column(String)
    name = Column(String)
    patronymic_name = Column(String)
    id_state = Column(Integer, ForeignKey('state.id'), nullable=False)
    last_modified = Column(DateTime, nullable=False, server_default="sysdate")
```

```
        dstaff = relationship("Staff", cascade="all")
        corporations = association_proxy('staff', 'corporation')
        @hybrid_property
        def full_name(self):
            return self.family_name + " " + self.name + " " + self.patronymic_name

    class Corporation(Base):
        _tablename_ = 'corporation'
        id = Column(Integer, primary_key=True)
        short_name = Column(String, nullable=False)
        full_name = Column(String, nullable=False, unique=True)
        last_modified = Column(DateTime, nullable=False, server_default="sysdate")
        dstaff = relationship("Staff", cascade="all")
        persons = association_proxy('staff', 'person')
    class Staff(Base):
        _tablename_ = 'staff'
        id = Column(Integer,
                primary_key=True)
        id_person = Column(Integer, ForeignKey('person.id'), nullable=False)
        id_corporation = Column(Integer, ForeignKey('corporation.id'), nullable=False)
        post = Column(String)
        last_modified = Column(DateTime, nullable=False, server_default="sysdate")
        person = relationship("Person", backref='staff')
        corporation = relationship("Corporation", backref='staff')
```

Для каждого из классов в атрибуте **_tablename_** указывается соответствующая таблица базы данных. Атрибуты, соответствующие полям таблицы, определяются через функцию **Column**. Следует обратить внимание, что для каждого класса задаются атрибуты, представляющие собой отношения с другими классами, через ключевое слово **relationship**. Кроме того, для класса **Person** задается ассоциация (association_proxy) с классом **Corporation** через таблицу **staff**. Для любого объекта класса **Person** можно получить связанные с ним объекты класса **Staff** через атрибут **dstaff**, а также непосредственно объекты класса **Corporation**, представляющие организации, где соответствующий человек работает, через атрибут corporations. Аналогично можно осуществлять навигацию для объектов всех других классов.

Для получения коллекции объектов классов **Person** и **Corporation** для тех организаций, где в названии содержится подстрока ‘Gis’, может быть выполнен следующий код:

```
for (p,c) in sess.query(Person,Corporation).\
                join(Staff).join(Corporation).\
                filter(Corporation.full_name.like('%Gis%')).\
                order_by(Corporation.full_name).\
                order_by(Person.full_name).\
                all():
    print(p,c)
```

Вместо использования текстовой строки SQL-запроса применяется вызов цепочки функций, таким образом, в рамках одного приложения достаточно использования только одного языка программирования. Нужно обратить внимание, что при выполнении соединений не указываются их условия, так как используется информация из определения атрибутов-отношений в классах.

Таким образом, можно выделить основные компоненты приложения, использующего объектно-реляционное отображение:

- структура классов приложения;
- отображение классов на таблицы базы данных;
- операции по выборке и модификации данных, над объектами классов, связанных с базой данных, для которых средствами ORM-библиотеки генерируется SQL-код.

4.3. Проблемы использования объектно-реляционного отображения

Несмотря на приведенные выше достоинства, существует ряд проблем, не позволяющих разрабатывать приложения исключительно с использованием объектно-реляционного отображения, без использования вызовов SQL-запросов в приложениях.

В том случае, если требуется не просто получить коллекцию объектов, удовлетворяющих некоему условию, получить данные, включающие сложные условия с группировкой, с использованием оконных функций и т.д., запись этих условий через функции ORM-библиотеки может оказаться гораздо более объемной и менее выразительной, чем непосредственный SQL-запрос. Кроме того, автоматически генерируемый SQL-код может содержать излишние соединения и приводить к неоптимальному выполнению.

Контрольные вопросы и упражнения

1. В чем заключается традиционный метод получения и сохранения данных в приложениях?
2. Назовите достоинства и недостатки традиционного метода.
3. Что представляет собой отношение в ORM?
4. Как выполняются соединения при использовании ORM?
5. Реализовать прототип приложения, моделирующего предметную область с использованием выбранной библиотеки ORM.
6. Перечислите основные недостатки ORM.

5. Многомерная модель данных

5.1. OLAP-технология

В области информационных технологий выделяют два класса информационными систем (и соответственно два класса задач):

- OLTP-системы;
- DSS-системы.

OLTP-системы (OnLine Transaction Processing) – системы оперативной обработки транзакций. Основная функция подобных систем заключается в одновременном выполнении большого числа коротких транзакций от большого числа пользователей.

Системы OLTP характеризуются:

- поддержкой большого числа пользователей;
- малым временем отклика на запрос;
- относительно короткими запросами;
- участием в запросах небольшого числа таблиц.

Бóльшая часть запросов OLTP-систем известна заранее еще на этапе проектирования системы. Критическими для OLTP-приложений являются скорость и надежность выполнения коротких запросов. Типичные примеры OLTP-систем: работа оператора в банке, ваши действия у банкомата.

Исторически такие системы возникли в первую очередь, поскольку реализовывали потребности в учете, скорости обслуживания, сборе данных и пр. Однако вскоре пришло понимание, что сбор данных – не самоцель, и накопленные данные могут быть полезны для извлечения информации.

Это привело к появлению другого типа систем (и соответственно класса задач) – ***систем поддержки принятия решений DSS*** **(*Decision***

Support System), ориентированных на анализ данных, на выполнение более сложных запросов, моделирование процессов предметной области, прогнозирование, нахождение зависимостей между данными (например, можно попытаться определить, как связан объем продаж товаров с характеристиками покупателей), для проведения анализа «что, если…».

DSS-системы оперируют с большими массивами данных, уже накопленными в OLTP-приложениях, взятыми их электронных таблиц или из других источников данных, и характеризуются следующими признаками:

- использование больших объемов данных;
- добавление в систему новых данных происходит относительно редко крупными блоками (например, раз в квартал загружаются данные по итогам квартальных продаж из OLTP-приложений);
- данные, добавленные в систему, обычно никогда не удаляются;
- перед загрузкой данные проходят различные процедуры «очистки», связанные с тем, что в одну систему могут поступать данные из многих источников, имеющих различные форматы представления, данные могут быть некорректны, ошибочны;
- небольшое число пользователей;
- очень часто новый запрос формулируется аналитиком для уточнения результата, полученного в результате предыдущего запроса (интерактивность);
- скорость выполнения запросов важна, но не критична.

Перечисленные характеристики требуют особой организации данных, отличных от тех, что используются в OLTP-системах (нормализованные реляционные таблицы).

Аналитические системы, ориентированные на аналитика, можно разделить:

- на статические DSS, известные в литературе как информационные системы руководителя (Executive Information Systems – EIS);
- динамические DSS.

EIS-системы содержат в себе предопределенные множества запросов (примером могут служить программные продукты финансового характера компании 1С).

Вторая группа (динамические DSS), напротив, ориентированы на обработку нерегламентированных запросов аналитиков к данным. Постепенно в этом направлении оформился ряд концепций хранения и анализа корпоративных данных:

- концепция хранилища данных (Data Warehouse);
- оперативная аналитическая обработка OLAP (OnLine Analytical Processing);
- интеллектуальный анализ данных (Data Mining).

Концепция хранилища данных определяет процесс сбора, отсеивания, предварительной обработки и накопления данных с целью:

- долговременного хранения данных;
- представления данных в форме, удобной для проведения анализа и создания аналитических отчетов.

Концепция OLAP – концепция комплексной оперативной обработки данных, использующая методы многомерного анализа данных в целях поддержки процессов принятия решений.

Концепция интеллектуального анализа данных определяет задачи поиска функциональных и логических закономерностей в накопленной информации, построение моделей и правил, которые объясняют найденные аномалии и/или прогнозируют развитие некоторых процессов.

OLAP-технология – это технология оперативной аналитической обработки данных, использующая методы и средства сбора, хранения и анализа многомерных данных в целях поддержки процессов принятия рещений. Автор реляционной модели данных Э. Кодд в 1993 г. сформулировал 12 принципов OLAP, которые позже были переработаны в так называемый тест FASMI:

- *Fast* (*быстрый*) – предоставление пользователю результатов анализа за приемлемое время;
- *Analysis* (*анализ*) – возможность осуществления любого логического и статистического анализа, характерного для данного приложения, и его сохранения в доступном для пользователя виде;
- *Shared* (*разделяемой*) – многопользовательский доступ к данным с поддержкой механизмов блокировок и средств авторизованного доступа;
- *Multidimensional* (*многомерной*) – многомерное концептуальное представление данных;

• *Information* (*информации*) – возможность обращаться к любой нужной информации независимо от ее объема и места хранения.

Реляционные СУБД, предназначенные для информационных систем оперативной обработки данных, в системах аналитической обработки показали себя несколько неповоротливыми и недостаточно гибкими. Более эффективными здесь оказываются многомерные СУБД. Многомерные СУБД являются узкоспециализированными СУБД, предназначенными для интерактивной аналитической обработки информации.

Основные понятия, используемые в этих СУБД: агрегируемость, историчность и прогнозируемость данных.

Агрегируемостъ данных означает рассмотрение информации на различных уровнях ее обобщения. В информационных системах степень детальности представления информации для пользователя зависит от его уровня: аналитик, пользователь-оператор, управляющий, руководитель.

Историчность данных предполагает обеспечение высокого уровня статичности (неизменности) собственно данных и их взаимосвязей, а также обязательность привязки данных ко времени. Статичность данных позволяет использовать при их обработке специализированные методы загрузки, хранения, индексации и выборки.

Временна́я привязка данных необходима для выполнения запросов, имеющих значения времени и даты в составе выборки. Необходимость упорядочения данных по времени в процессе обработки и представления данных пользователю накладывает требования на механизмы хранения и доступа к данным. Так, для уменьшения времени обработки запросов желательно, чтобы данные всегда были отсортированы в том порядке, в котором они наиболее часто запрашиваются.

Прогнозируемость данных подразумевает задание функций прогнозирования и применение их к различным временны́м интервалам.

5.2. Концептуальная модель данных

Многомерный анализ – анализ по нескольким независимым измерениям, вдоль которых могут быть проанализированы определенные совокупности данных. OLAP-технология представляет для анализа данные в виде многомерных наборов данных, называемых многомер-

ными кубами (гиперкуб, метакуб, куб фактов). В основе многомерного представления данных лежит их разделение на две группы – измерения и факты (меры). Осями многомерной системы координат служат основные атрибуты анализируемого бизнес-процесса (то, по чему ведется анализ). Например, для продаж это могут быть тип товара, регион, тип покупателя. В качестве одного из измерений используется время. На пересечениях осей-измерений (dimensions) находятся данные, количественно характеризующие процесс, – факты (меры, measures): суммы и иные агрегатные функции: min, max, avg, дисперсия, среднее отклонение и пр. (рис. 5.1).

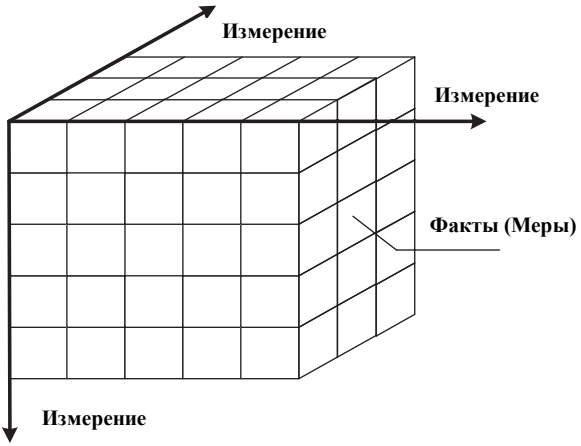


Рис. 5.1. Представление данных в виде гиперкуба

Измерение (*dimension*) – множество однотипных данных, образующих одну из граней гиперкуба. Примерами наиболее часто используемых временных измерений являются Дни, Месяцы, Кварталы и Годы. В качестве географических измерений широко употребляются Города, Районы, Регионы и Страны. В многомерной модели данных измерения играют роль индексов, служащих для идентификации конкретных значений в ячейках гиперкуба. Измерения могут быть и числовыми, если какой-либо категории (например, наименованию товара) соответствует числовой код, но в любом случае это данные дискретные, т. е. принимающие значения из ограниченного набора. Измерения качественно

описывают исследуемый бизнес-процесс. На рис. 5.2 приведены примеры измерений и направлений консолидации данных.

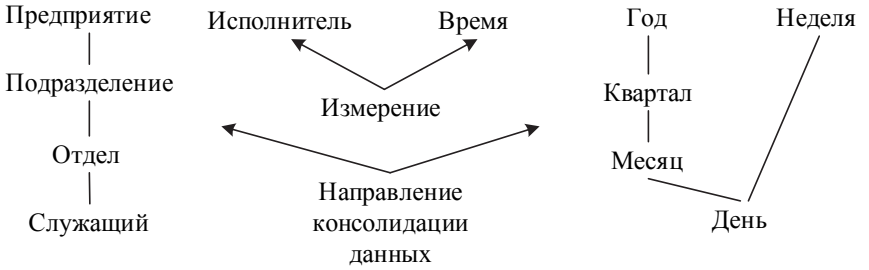


Рис. 5.2. Примеры измерений и направлений консолидации данных

Факты (*measure, синонимы: показатель, ячейка, мера*) – это данные, количественно описывающие бизнес-процесс, непрерывные по своему характеру, определяемые набором измерений. Примеры фактов – цена товара или изделия, их количество, сумма продаж или закупок, зарплата сотрудников, сумма кредита, страховое вознаграждение и т.д.

Многомерный куб можно рассматривать как систему координат, осями которой являются измерения, например, «Дата», «Товар», «Покупатель». По осям будут откладываться значения измерений – даты, наименования товаров, названия фирм-покупателей, ФИО физических лиц и т.д.

В такой системе каждому набору значений измерений (например, «дата – товар – покупатель») будет соответствовать ячейка, в которой можно разместить числовые показатели (т. е. факты), связанные с данным набором. Таким образом, между объектами бизнес-процесса и их числовыми характеристиками будет установлена однозначная связь. Принцип организации многомерного куба на примере работы торговой организации поясняется на рис. 5.3.

В ячейке 1 будут располагаться факты, относящиеся к продаже цемента ООО «Спецстрой» в марте, в ячейке 2 – к продаже плит ЗАО «Пирамида» в июне, а в ячейке 3 – к продаже плит ООО «Спецстрой» в апреле 2018 года.

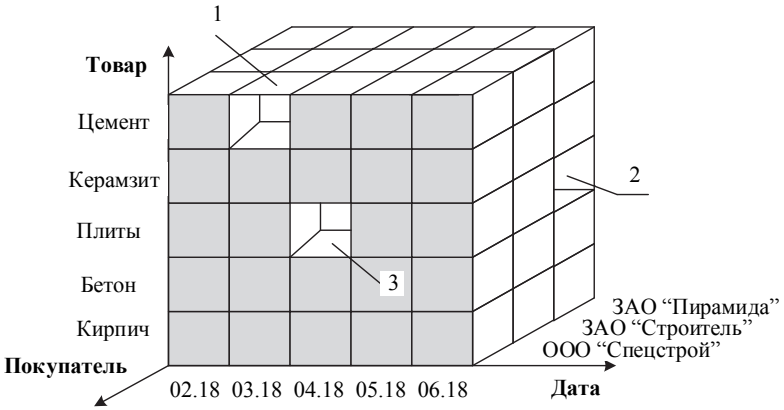


Рис. 5.3. Принцип организации многомерного куба

Многомерный взгляд на измерения «Дата», «Товар», «Покупатель» представлен на рис. 5.4.

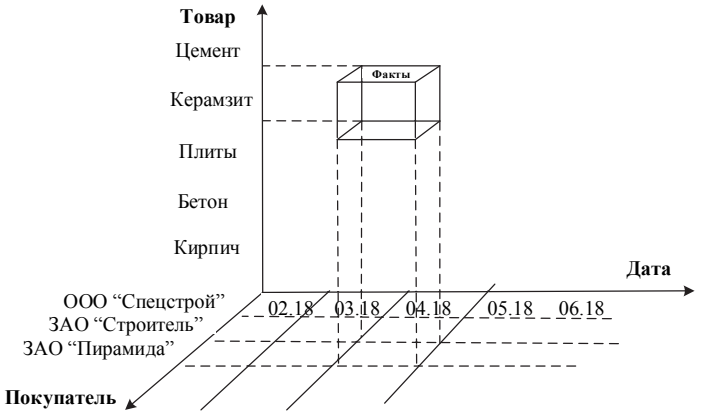


Рис. 5.4. Измерения и факты в многомерном кубе

Фактами в данном случае могут быть «Цена», «Сумма», «Количество». Тогда выделенный сегмент будет содержать информацию о том, сколько керамзита, на какую сумму и по какой цене приобрела компания «Пирамида» в апреле месяце 2018 года.

В существующих СУБД с многомерной моделью используются два основных варианта (схемы) организации данных: *гиперкубическая* и *поликубическая*. В поликубической схеме предполагается, что в базе данных может быть определено несколько гиперкубов с различной размерностью и с различными измерениями в качестве граней. В случае гиперкубической схемы предполагается, что все показатели определяются одним и тем же набором измерений. Очевидно, в некоторых случаях информация в базе может быть избыточной (если требовать обязательное заполнение ячеек).

Если речь идет о многомерной модели с размерностью больше двух, то не обязательно визуально информация представляется в виде многомерных объектов (трех-, четырех- и более мерных гиперкубов). Пользователю и в этих случаях более удобно иметь дело с двухмерными таблицами или графиками. Данные при этом представляют собой «вырезки» (точнее, «срезы») из многомерного хранилища данных, выполненные с разной степенью детализации.

Следует отметить, что гиперкуб является лишь концептуальной логической моделью организации данных, а не физической реализацией их хранения, поскольку храниться такие данные могут и в реляционных таблицах.

В самом общем виде

OLAP = многомерное представление = Куб

В понятии многомерности можно выделить три уровня.

- *Многомерное представление данных* – средства, обеспечивающие многомерную визуализацию и манипулирование данными; этот слой многомерного представления абстрагирован от физической структуры данных и воспринимает данные как многомерные.
- *Многомерная обработка* – средство (язык) формулирования многомерных запросов и процессор, умеющий обработать и выполнить такой запрос.
- *Многомерное хранение* – средства физической организации данных, обеспечивающие эффективное выполнение многомерных запросов.

Первые два уровня в обязательном порядке присутствуют во всех OLAP-средствах. Третий уровень, хотя и является широко распростра-

ненным, не обязателен, так как данные для многомерного представления могут извлекаться и из реляционных структур.

При этом следует отметить, что многомерность модели данных означает прежде всего многомерное логическое представление структуры информации при описании и в операциях манипулирования данными, а уж затем многомерную визуализацию цифровых данных.

По сравнению с реляционной моделью многомерная организация данных обладает более высокой наглядностью и информативностью. Для иллюстрации на рис. 5.5 приведены реляционное (*а*) и многомерное (*б*) представление одних и тех же данных об объемах продаж автомобилей.

Модель	Месяц	Объем
ВАЗ Калина	апрель	10
ВАЗ Калина	май	15
ВАЗ Калина	июнь	4
Ford focus	апрель	10
Ford focus	июнь	18
Renoult duster	июнь	20
Scoda rapid	апрель	5

а

Модель	Апрель	Май	Июнь
ВАЗ Калина	10	15	4
Ford focus	10	N	18
Renoult duster	N	N	20
Scoda rapid	5	N	N

б

Рис. 5.5. Реляционное (*а*) и многомерное представление данных (*б*)

В примере на рис. 5.5, *б* каждое значение ячейки «Объем продаж» однозначно определяется комбинацией временного измерения (месяц продаж) и модели автомобиля. На практике часто требуется большее количество измерений.

Пример трехмерной модели данных приведен на рис. 5.6. Символами «A», «B», «C» условно обозначены регионы продаж.

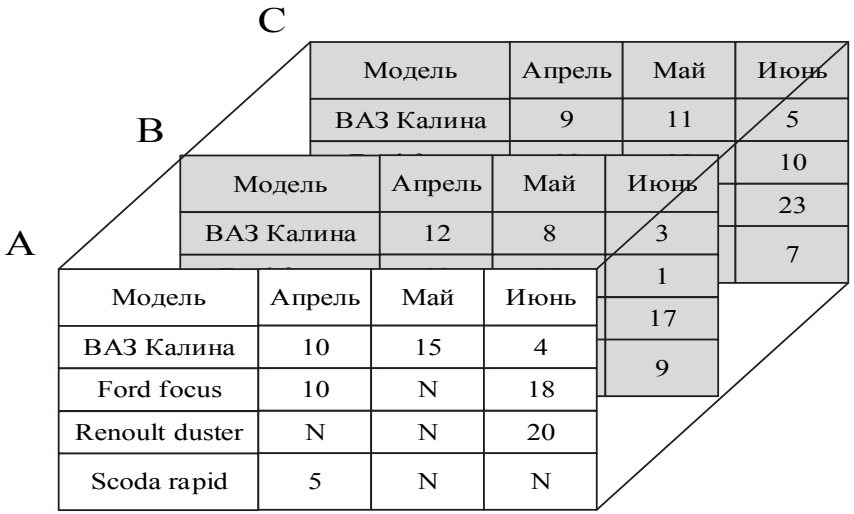


Рис. 5.6. Трехмерная модель данных

В примере на рис. 5.7 показан трехмерный куб, где в качестве фактов использованы суммы продаж, а в качестве измерений – время, товар и магазин, определенные на разных уровнях группировки: товары группируются по категориям, магазины – по странам, а данные о времени совершения операций – по месяцам.

Рис. 5.7. Анализ данных продаж товаров

При использовании многомерной модели данных в OLAP-технологии применяется ряд специальных аналитических операций, к которым относятся: формирование «среза», «вращение», агрегация и детализация.

- Операция *Сечения* (*срез – Slice*): формируется подмножество гиперкуба, в котором значение одного или более измерений фиксировано (значение параметров для фиксированного, например, месяца). Например, если ограничить значения измерения «Модель автомобиля» в гиперкубе (рис. 5.6) маркой «Калина», то получится двухмерная таблица продаж этой марки автомобиля различными дилерами по годам. На рис. 5.8 схематично представлено сечение гиперкуба: слева сечение выполнено при фиксированном значении измерения «Дата», и полученный срез содержит информацию обо всех товарах и всех покупателей за определенный месяц, на правом фрагменте рисунка получено два среза, пересечение которых будет содержать информацию обо всех покупателях, но на определенный товар и за определенный месяц.

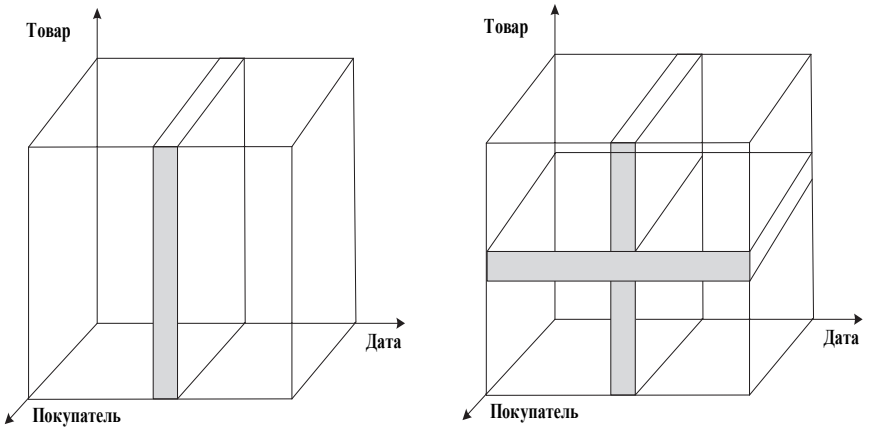


Рис. 5.8. Операция сечения гиперкуба

- Операция *Вращения* (*Rotate*): меняется порядок представления измерений при визуальном представлении данных, обеспечивая представление гиперкуба в более удобной для восприятия форме (рис. 5.9).

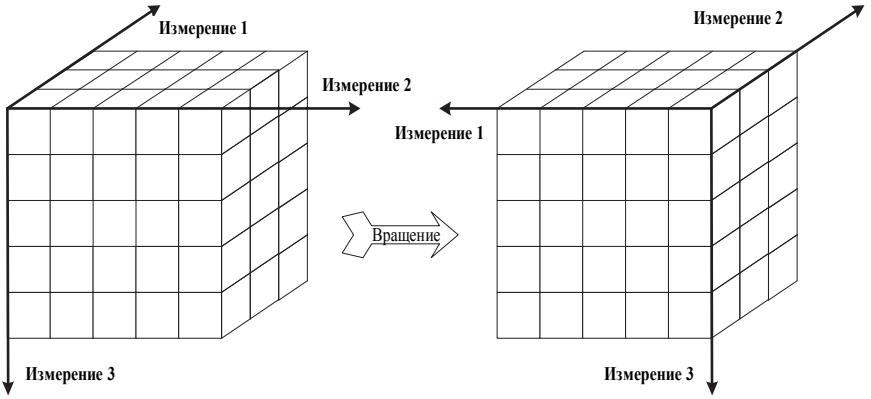


Рис. 5.9. Операция вращения гиперкуба

Так, например, «вращение» двумерной таблицы, показанной на рис. 5.5, *б*, приведет к изменению ее вида таким образом, что по оси X будет марка автомобиля, а по оси Y – время.

- Операция *Агрегации* (*консолидации – Drill up*) – операция, которая определяет переход по направлению от детального (down) представления данных к агрегированному (up). Направление обобщения может быть задано как по иерархии отдельных измерения, так и согласно прочим отношениям, установленным в рамках измерений или между измерениями. Например, показатели для отдельных компаний могут быть просто просуммированы с целью получения суммарных показателей для каждого города, а показатели для городов могут быть «свернуты» до показателей по отдельным странам.
- Операция *Детализации* (*спуска – Drill doun*) – операция, обратная операции консолидации, которая определяет переход по направлению от агрегированного (up) представления данных к детальному (down) и включает отображение подробных сведений для рассматриваемых консолидированных данных.

Схематично операции консолидации и детализации представлены на (рис. 5.10).

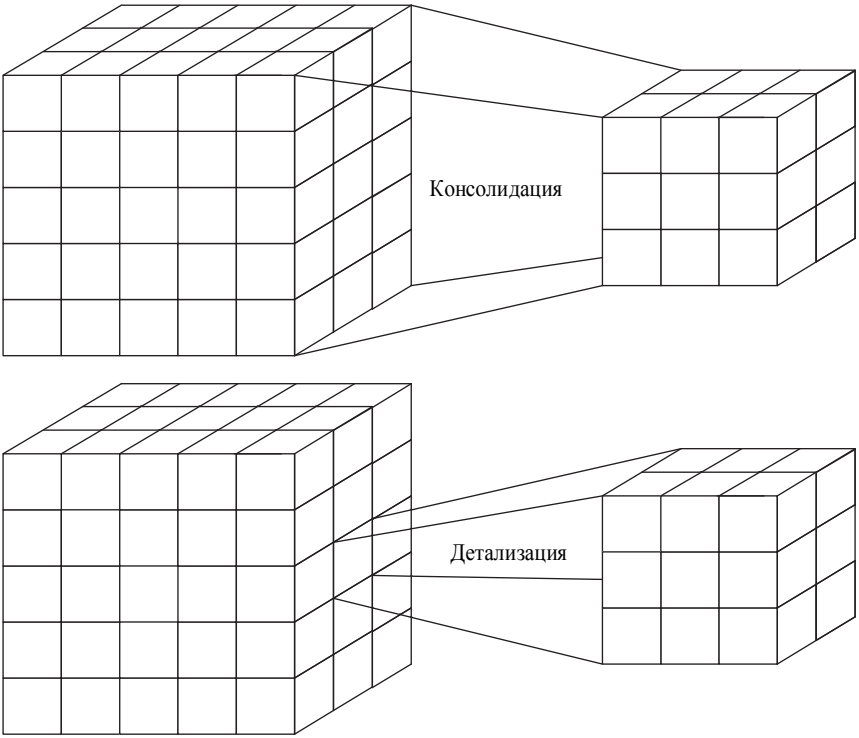


Рис. 5.10. Операция консолидации и детализации

- Операция *Разбиение с поворотом* (*Slicing and dicing*) позволяет получить представление данных с разных точек зрения; например, один срез данных о доходах может содержать все сведения о доходах от продаж товаров указанного типа по каждому городу, другой срез может представлять данные о доходах отдельной компании в каждом из городов.

Перечисленные операции часто используются в OLAP-системах анализа данных. Так, например, при выборе отдельного факультета (рис. 5.11, *а*) мы получаем данные по кафедрам этого факультета (рис. 5.11, *б*), а при последующем выборе отдельной кафедры мы получаем подробные данные по данной кафедре (рис. 5.11, *в*).

Факультет	Основные	Совместители	Внут. совместители	Почасовики	Все ППС
АВТФ	54,7	51,6	58,5		54,0
МТФ	48,3	53,3	52,0		49,3
РЭФ	55,2	53,5	59,8		55,0
ФБ	46,9	48,7	60,8	43,3	47,6
ФГО	43,2	49,1	46,5	37,6	43,9
ФЛА	51,4	54,1	58,2		53,0
ФМА	52,7	50,7	63,0		52,6
ФПМИ	49,9	50,0	51,9	52,3	50,3
ФТФ	54,8	57,4	51,0	35,3	55,6
ФЭН	52,8	47,1	63,0	55,0	51,5
ЮФ	40,8	44,7	38,0	29,0	42,7
Общеуниверситетские	50,7	59,0	66,0		51,1
ИДПО	52,5	40,0			47,5
Итого по университету	49,7	51,2	55,1	44,8	50,1

а

Факультет	Основные	Совместители	Внут. совместители	Почасовики	Все ППС
АиМЛ	54,8	39,8			50,2
ВМ	51,5	58,0			51,7
Выч.технол.	48,3	55,0		61,0	57,5
ИМ	54,8	51,2			53,9
ПВТ		51,9			51,9
ПМт	42,2	52,0	51,5	31,4	43,1
ПСиБД	44,0	48,8	53,0		46,4
Итого по университету	49,9	50,0	51,9	52,3	50,3

б

Факультет	Основные	Совместители	Внут. совместители	Почасовики	Все ППС
АБРАМОВ М.В.		29,0			29,0
БЕКАРЕВА Н.Д.	68,0				68,0
БЕЛОВА Т.И.			48,0		48,0
БЕРДНИКОВ В.С.		65,0			65,0
ДЕНИСОВ В.И.			75,0		75,0
ДОМНИКОВ П.А.	27,0				27,0
ЕЛАНЦЕВА И.Л.	41,0				41,0
ЗАДОРОЖНЫЙ А.Г.	34,0				34,0
ЗЕНКОВА Н.Ю.				41,0	41,0
ИГНАТЬЕВ А.Н.				27,0	27,0

в

Рис. 5.11. Операция спуска (Drill doun) в OLAP-системе анализа данных:

а – средний возраст профессорско-преподавательского состава по факультетам; *б* – средний возраст профессорско-преподавательского состава по кафедрам факультета; *в* – средний возраст профессорско-преподавательского состава по кафедре

5.3. Реализация многомерной модели данных

Многомерный гиперкуб, используемый в OLAP-технологии, может быть реализован в рамках реляционной модели или существовать как отдельная база данных специальной многомерной структуры. В зависимости от этого принято различать реляционный (ROLAP-модель) и многомерный (MOLAP-модель) подходы к построению модели данных.

В MOLAP-модели многомерное представление данных реализуется физически – не в форме реляционных таблиц, а в виде упорядоченных многомерных массивов.

Достоинства MOLAP-модели

- Высокая производительность. Поиск и выборка данных осуществляется значительно быстрее, чем при многомерном концептуальном взгляде на реляционную базу данных, так как многомерная база данных денормализована, содержит заранее агрегированные показатели и обеспечивает оптимизированный доступ к запрашиваемым ячейкам.
- Легко добавляются разнообразные встроенные функции, в случае реляционных СУБД это сделать сложнее.

Недостатки MOLAP-модели

- Многомерные СУБД по сравнению с реляционными очень неэффективно используют внешнюю память – в подавляющем большинстве случаев многомерный гиперкуб является сильно разреженным.
- Многомерные СУБД не позволяют работать с большими объемами данных.

Из сказанного следует, что использование многомерных СУБД оправданно только при следующих условиях.

1. Объем исходных данных для анализа не слишком велик (не более нескольких гигабайт).

2. Набор информационных измерений стабилен, так как любое изменение в их структуре почти всегда требует полной перестройки гиперкуба.

3. Время ответа системы на запросы является наиболее критичным параметром.

Примеры OLAP-серверов, использующих MOLAP-архитектуру: Oracle OLAP Option, IBM Informix MetaCube, IBM DB2 OLAP, Microsoft Analysis Services, Arbor Essbase и др.

ROLAP-модель предполагает хранение данные в реляционной базе данных специальной структуры, а после преобразование через промежуточный слой метаданных данные отображаются в многомерной форме. Иначе говоря, гиперкуб эмулируется СУБД на логическом уровне.

Примеры OLAP-серверов, использующих ROLAP-архитектуру: те же: Oracle OLAP Option, Microsoft Analysis Services, Arbor Essbase, а также IBM Informix Red Brick, HighGate Project компании Sybase и др.

В ROLAP-модели наиболее эффективным способом моделирования многомерного куба фактов является ***схема «звезда» (star schema).***

Основными составляющими структуры данных ROLAP-модели являются таблица фактов (fact table) и таблицы измерений (dimension tables).

Таблица фактов является основной таблицей данных и содержит сведения об объектах или событиях, совокупность которых будет в дальнейшем анализироваться. Одна строка таблицы фактов – одна ячейка гиперкуба.

Обычно говорят о четырех наиболее часто встречающихся типах фактов:

- факты, связанные с транзакциями (Transaction facts); основаны на отдельных событиях (типичными примерами является телефонный звонок или снятие денег со счета с помощью банкомата);
- факты, связанные с «моментальными снимками» (Snapshot facts); основаны на состоянии объекта (например, банковского счета) в определенные моменты времени, например, на конец дня или месяца; типичные примеры: объем продаж за день или дневная выручка;
- факты, связанные с элементами документа (Line-item facts); основаны на том или ином документе (например, счете за товар или услуги) и содержат подробную информацию об элементах этого документа (например, количество, цена, процент скидки);
- факты, связанные с событиями или состоянием объекта (Event or state facts); представляют возникновение события без подробностей о нем (например, просто факт продажи или факт отсутствия таковой без иных подробностей).

Таблица фактов индексируется по сложному ключу, составленному из ключей отдельных изменений. Помимо этого, таблица фактов

содержит одно или несколько числовых полей, на основании которых в дальнейшем будут получены агрегатные данные.

Таблица измерений содержит неизменяемые или редко изменяемые данные. В каждой таблице измерений перечислены значения одного из измерений гиперкуба. Таблицы измерений содержат целочисленное ключевое поле (обычно это суррогатный ключ) для идентификации члена измерения и, как минимум, одно описательное поле. Каждая таблица измерений находится в отношении «один ко многим» с таблицей фактов (рис. 5.12).

Рис. 5.12. Схема «звезда»

Причина, по которой данная схема названа «звездой» достаточно очевидна: таблица фактов расположена в центре, а таблицы измерений образуют лучи. При этом информация о каждом измерении располагается в отдельной таблице, что делает схему логически прозрачной и понятной пользователю.

В более сложных задачах используются различные расширения схемы «звезда» – ***схема «снежинка» (snowflake schema).*** Это расширение может проявляться в двух разновидностях.

В случае большого числа сложных атрибутов в таблице измерений некоторые атрибуты могут быть детализированы в отдельных дополнительных таблицах измерений. В этом случае отдельные измерения содержатся не в одной, а в нескольких связанных между собой таблицах (рис. 5.13). Это не только уменьшает избыточность при хранении, но и исключает возникновение противоречий, если, например, один товар ошибочно отнесут к разным группам.

При большом числе независимых измерений бывает полезным поддерживать множество таблиц фактов, соответствующих каждому возможному сочетанию выбранных в запросе измерений. Это, с одной стороны, позволяет добиться лучшей производительности, с другой стороны, приводит к избыточности данных и усложнениям в структуре базы данных, в которой оказывается огромное количество таблиц фактов.

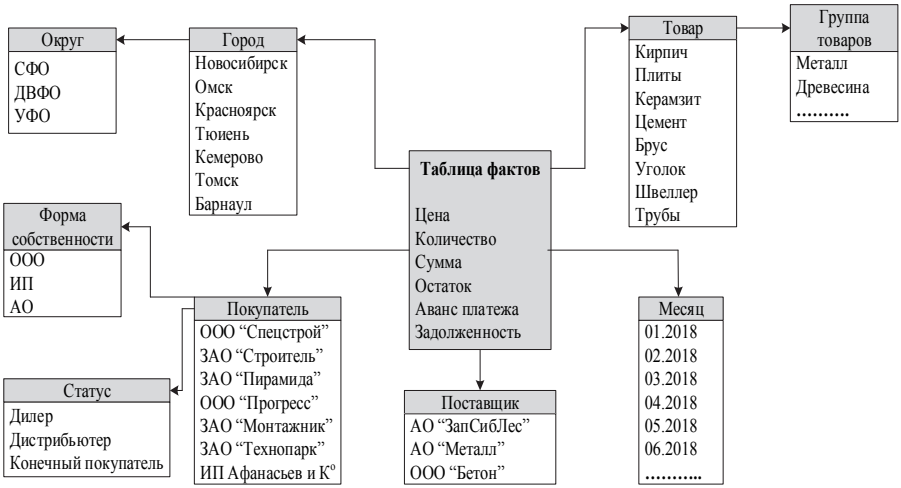


Рис. 5.13. Схема «снежинка»

Ниже приведен пример формирования многомерного куба в ROLAP-модели. Записанный запрос решает задачу получения средних объемов продаж товаров каждого поставщика с разбивкой по покупателям и по месяцам.

select avg(количество), поставщик, покупатель, месяц

from таблица фактов, таблица поставщиков, таблица покупателей

where таблица фактов.код_поставщика= таблица поставщиков.код_поставщика

and таблица фактов. код_покупателя = таблица покупателей.код_покупателя

group by поставщик, покупатель, месяц

order by поставщик, покупатель, месяц

Достоинства использования реляционных баз данных в OLAP-системах

1. При использовании ROLAP-модели размер базы данных не является таким критичным фактором, как в MOLAP-модели.

2. Внесение изменений в структуру измерений не требует физической реорганизации базы данных, как в случае MOLAP-модели.

Главный недостаток ROLAP-модели по сравнению с MOLAP-моделью – меньшая производительность.

Гибридная модель (Hybrid OLAP, HOLAP) совмещает достоинства и минимизирует недостатки, присущие предыдущим моделям (Media/MR компании Speedware). Исходные данные остаются в реляционной базе данных, а агрегатные данные хранятся в многомерной базе данных.

5.4. Расширения языка SQL для OLAP-анализа данных

Язык SQL обладает достаточными возможностями для выполнения аналитических отчетов. Но у языка SQL есть следующие ограничения.

- *Сортировка данных*. Многие аналитические запросы явно или неявно требуют предварительной сортировки данных (первые 10 %, самые низкоприбыльные и т.п.). Однако SQL оперирует с несортированными строками. Единственным средством сортировки является фраза **order by** в операторе **select**, причем сортировка выполняется в самом конце процесса, когда данные уже отобраны.
- *Хронологические последовательности*. Многие запросы к хранилищам данных предназначены для анализа изменения некоторых показателей во времени (сравнение результатов одного года с другим,

одного месяца некоторого года с результатами некоторого месяца другого года). Средствами SQL это зачастую сделать весьма трудно.

- *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 SQL-запрос трудно выражаем средствами SQL. А сводный отчет классического формата – с детальными данными, промежуточными и сводными итогами – единым SQL-запросом вообще не выразить, поскольку структура всех строк запроса должна быть одинаковой.

В связи с этим разработчики OLAP-средств предлагают расширения языка SQL для аналитической обработки. Например, в языке RISQL (Red Brick Intelligent SQL) СУБД Red Brick компании IBM введены следующие расширения:

- *диапазоны* – позволяют сформировать запросы типа «отобрать первые 10 записей»;
- *перемещение итогов и средних* – используются для хронологического анализа, требующего предварительной обработки данных;
- *расчет текущих итогов и средних* – позволяют получать, например, данные по отдельным месяцам, а также годовой итог на текущую дату;
- *сравнительные коэффициенты* – позволяют создавать запросы, выражающие отношения отдельных значений к общим и промежуточным итогам без использования сложных вложенных запросов;
- *промежуточные итоги* – позволяют получать результаты запросов, в которых объединена детализированная и итоговая информация.

Ниже приведено несколько примеров запросов на языке RISQL.

1. Получить значения ежеквартальных объемов продаж для отделения компании, указав общий итог с начала года и до текущего месяца.

В приведенном ниже запросе используется таблица

Branch_sales (отделение_компании, квартал, объем_продаж_за_квартал)

и функция **cume** для вычисления текущего и общего итога для некоторого столбца:

```
select квартал, объем_продаж_за_квартал,
    cume(объем_продаж_за_квартал) as Year-To-Date
    from Branch_sales where отделение_компании = “B3”;
```

Результатом запроса является следующая таблица.

Квартал	Ежеквартальный объем продаж	Суммарный объем продаж с начала года
1	960 000	960 000
2	1 290 000	2 250 000
3	2 000 000	4 250 000
4	1 500 000	5 750 000

2. Получить значения ежемесячных объемов продаж для отделения компании для первых шести месяцев без учета сезонной составляющей.

Решение задачи достигается использованием функции **movingavg**, вычисляющей трехмесячное скользящее среднее значение и функции **movingsum**, вычисляющее трехмесячное скользящее суммарное значение на основе текущей и (n – 1) предыдущих строк, что позволит исключить сезонную составляющую.

select месяц, объем_продаж_за_месяц,

movingavg (объем_продаж_за_месяц) as 3-Month-Moving-Avg,

movingsum (объем_продаж_за_месяц) as 3-Month-Moving-Sum

from Branch_sales where отделение_компании = “B3”;

Результатом запроса является следующая таблица.

Месяц	Ежемесячный объем продаж	3-месячное скользящее среднее значение	3-месячное скользящее суммарное значение
1	210 000	–	–
2	350 000	–	–
3	400 000	320 000	960 000
4	420 000	390 000	1 170 000
5	440 000	420 000	1 260 000
6	430 000	430 000	1 290 000

Контрольные вопросы и упражнения

1. Перечислите основные характеристики OLTP-систем.
2. Перечислите основные характеристики OLAP-систем.
3. В чем состоит тест FASMI?
4. Каковы основные понятия многомерной модели данных? Дайте им характеристику.
5. В чем отличие гиперкубической и поликубической организаций данных?
6. Что вкладывается в понятие многомерности в OLAP-анализе?
7. Перечислите и дайте характеристику аналитическим операциям многомерной модели данных.
8. Каковы достоинства и недостатки MOLAP-модели данных?
9. Каковы достоинства и недостатки ROLAP-модели данных?
10. Каково назначение и структура таблицы фактов в ROLAP-модели?
11. Каково назначение и структура таблицы измерений в ROLAP-модели?
12. Какова организация данных в схеме «звезда»?
13. Какова организация данных в схеме «снежинка»?
14. В чем состоят расширения языка SQL для OLAP-анализа данных?

6. NoSQL базы данных

6.1. NoSQL. Общие положения

Термин NoSQL может трактоваться и как отрицание применения SQL как языка доступа к данным, так и как возможность использования альтернативных методов доступа (Not Only SQL).

Стандартная архитектура информационных систем, сложившаяся к 2000-м годам, – единая база данных обрабатываемой СУБД на вертикально-масштабируемом сервере, либо кластере из небольшого числа также вертикально-масштабируемых серверов с размещением базы данных на общем хранилище. При этом с базой данных взаимодействует несколько приложений, обрабатывающих данные.

В рамках парадигмы NoSQL происходит переход к множеству баз данных, каждая из которых предназначена для отдельного приложения. Данные, относящиеся к одному и тому же объекту предметной области, могут находиться в различных базах данных. Каждая база данных может быть разделена по узлам кластера, при этом число узлов в кластере может быть сколь угодно большим. Итак, основополагающее отличие: реляционная база данных: много приложений – одна база данных, NoSQL: много приложений – много баз данных.

NoSQL-модели данных разделяют на четыре типа:

- Ключ-значение;
- Столбцовые;
- Документные;
- Графовые.

Первые три типа относят к агрегатно-ориентированным моделям данных. Это означает, что единицей логической структуры базы дан-

ных является агрегат – объект, непосредственно используемый в приложении, работающем с базой данных.

Графовая модель данных основана на представлении данных как атрибутов вершин и ребер ориентированного графа. Такая модель позволяет выполнять запросы, являющиеся по своей сути поиском маршрута на графе.

В отличие от реляционной модели данных, которая может рассматриваться безотносительно конкретных реляционных СУБД, для понимания NoSQL необходимо знакомиться с примерами работы в конкретных NoSQL СУБД, а также с основами их архитектуры, в особенности, с реализацией согласованности и высокой доступности данных. Для этого будем рассматривать NoSQL модели данных на примере их реализации в соответствии с приведенной таблицей.

Модель	**СУБД**
Ключ-значение	Redis
Столбцовая	Cassandra
Документная	MongoDB
Графовая	Neo4j

Далее в этом разделе, до непосредственного рассмотрения графовой модели данных, при упоминании NoSQL будут подразумеваться именно агрегатно-ориентированные модели данных (ключ-значение, столбцовая, документная).

Перед подробным представлением каждой из моделей рассмотрим для перечисленных выше СУБД (Redis, Cassnadra, MongoDB) такие аспекты:

- возможные операции выборки данных;
- распределение и репликация данных между узлами;
- обеспечение согласованности данных.

Выборка данных в NoSQL

Этот аспект характеризует ключевое отличие от реляционных СУБД и по сути объясняет само название NoSQL – в NoSQL СУБД не существует декларативного языка для выборки данных. Даже синтаксически схожий с SQL язык запросов CQL в Cassandra не является

декларативным. Рассмотрим кратко возможности выборки для каждой из моделей.

Модель *Ключ-значение*

Выборка отдельных объектов: получение значения по ключу / множества значений по множеству ключей. Общей схемы данных для коллекции нет. Значением, возвращаемым по ключу, может быть некий объект со структурой, не интерпретируемой на уровне базы данных, либо базовая структура данных, содержащая объекты (массив, словарь, очередь, стек).

Остальные операции, знакомые по работе с SQL (отбор по условию на поля, соединения, группировка), отсутствуют.

Модель *Столбцовой базы данных*

В столбцовой базе данных для каждого из семейства столбцов (можно с определенными допущениями считать аналогом таблицы в реляционной базе данных) заранее задается схема, с отдельными атрибутами, часть из которых должна составлять ключ. Помимо выбора строк по условию на ключевые атрибуты, возможна также выборка по условиям на неключевые, при выполнении ряда ограничений. Операции соединения для семейств столбцов не существует. Группировка данных по подмножеству ключевых атрибутов возможна с рядом ограничений.

Модель *Документной базы данных*

Схема для коллекции не задается заранее, но все добавляемые объекты – **документы** имеют свою структуру, интерпретируемую на уровне базы данных. Возможна выборка документов как по ключу, так и по условию на элементы их структуры. Для выполнения соединений и группировок существуют отдельные функции, при этом выполнение соединений на уровне как базы данных, так и приложения является нежелательным, так как характеризуется низкой производительностью.

Как можно было понять, функциональность операции выборки данных в NoSQL существенно уступает таковой в реляционных СУБД, однако такая ограниченность предназначена для достижения высокой скорости отклика при выполнении любых допустимых запросов для любого объема данных, в том числе и распределенных по множеству узлов. Отсюда следует и еще одна принципиальная разница между

NoSQL моделью данных и реляционной – в подходе к проектированию базы данных.

Для проектирования реляционной базы данных в первую очередь выполняется анализ данных для получения нормализованной структуры отношений, не допускающей потери согласованности при операциях модификации данных. После реализации спроектированных отношений как таблиц возможна реализация сколь угодно большого числа произвольных запросов к ним. Однако при этом невозможно в общем случае обеспечить заданное время отклика для них для сколь угодно больших объемов данных.

При проектировании NoSQL базы данных анализ начинается с определения перечня всех возможных запросов (особенно характерно для Cassandra), а уже исходя из них реализуются структуры данных, содержащих агрегаты (коллекции, семейства столбцов – в зависимости от типа базы данных). Как правило, данные будут храниться в денормализованном виде, а также дублироваться в различных структурах одной и той же базы данных. В самой СУБД не существует механизма обеспечения согласованности между агрегатами, хранящимися в разных структурах при модификации данных. Такая согласованность должна обеспечиваться приложением, работающим с базой данных.

Распределение и репликация данных между узлами

Необходимость обеспечить возможность эффективной обработки данных сколь угодно большого объема, распределенных по сколь угодно большому числу узлов, собственно, и привело к возникновению концепции NoSQL с ее функциональными ограничениями операции выборки и отказу от нормализации хранимых данных. В том случае, если основной способ получения данных – это получение агрегата по ключу и не требуется выполнять соединений, то алгоритм выборки данных можно представить следующим образом.

1. Для значения ключа s вычислить хэш-функцию n = f(s); f(строка)->[1..N], где N – общее количество узлов

2. В узле по ключу из соответствующей структуры данных (хэш-таблица или бинарное дерево поиска) извлекается значение агрегата.

Таким образом, выполняется обращение только к одному узлу кластера.

Для исключения операций чтения с диска для отсутствующего по ключу значения выполняется вычисление фильтра Блума, и, если результат равен 0, то это однозначно свидетельствует о том, что по данному ключу значений в таблице нет. Если результат равен 1, то значение существует с некоей вероятностью и обращаться к диску нужно. В данном учебном пособии не будет рассматриваться непосредственная реализация на уровне СУБД (фильтр Блума, Memtable, SSTable).

Репликация осуществляется для обеспечения доступности данных, при выходе узла из строя. Количество узлов, на которые записываются реплики агрегатов, определяется в конфигурации конкретной базы данных.

Обеспечение согласованности данных

Свойства ACID, как правило, обеспечиваются NoSQL СУБД только на уровне отдельного агрегата, так как транзакции, затрагивающие несколько агрегатов, в общем случае являются распределенными. Для распределенных баз данных (к которым относятся NoSQL базы данных) было сформулировано следующее утверждение, ставшее известным как **теорема CAP**.

Теорема CAP. В распределенной системе из трех характеристик согласованность (C), доступность (A), устойчивость к разделению сети (P) могут выполняться только две.

Эти утверждения могут быть проиллюстрированы на рис. 6.1–6.6.

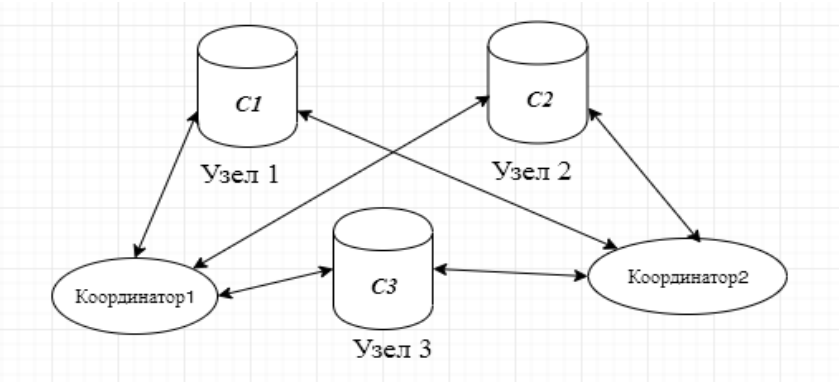


Рис. 6.1. Общая схема распределенной базы данных

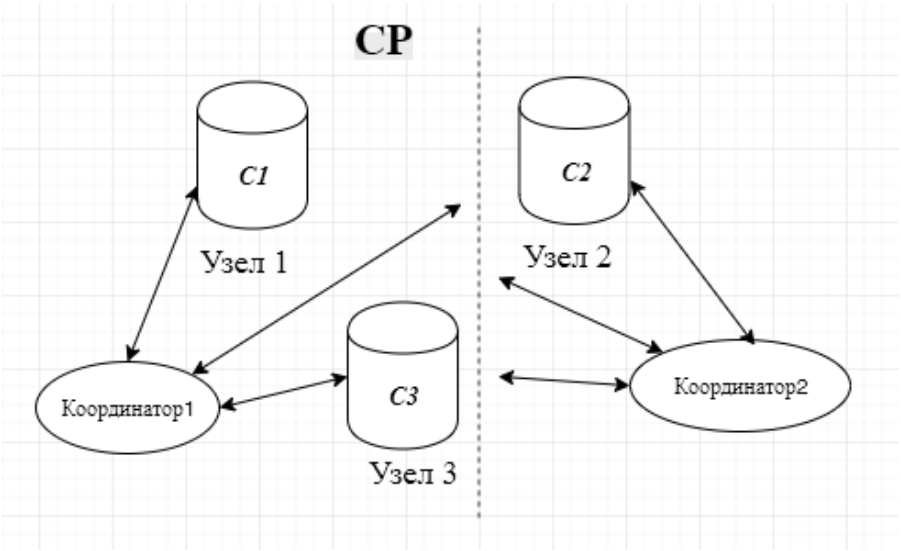


Рис. 6.2. Выполнение свойств CP

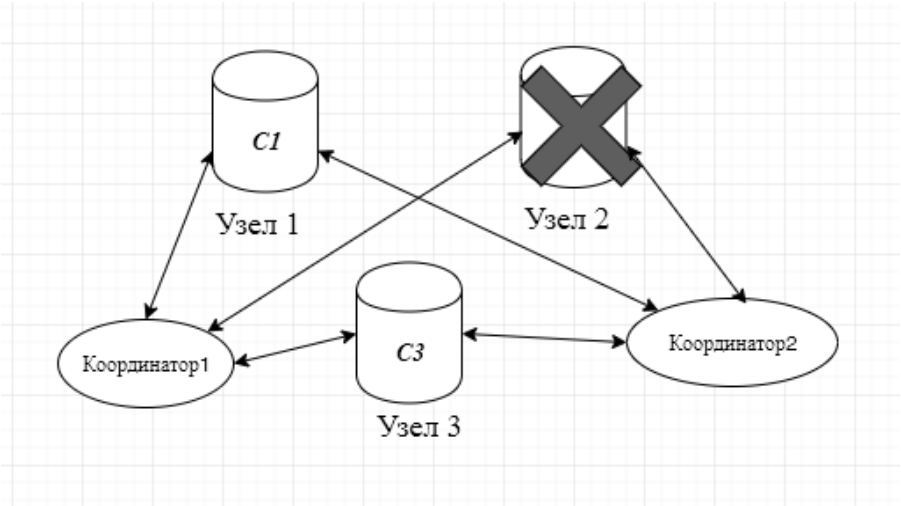


Рис. 6.3. Невыполнение свойства A

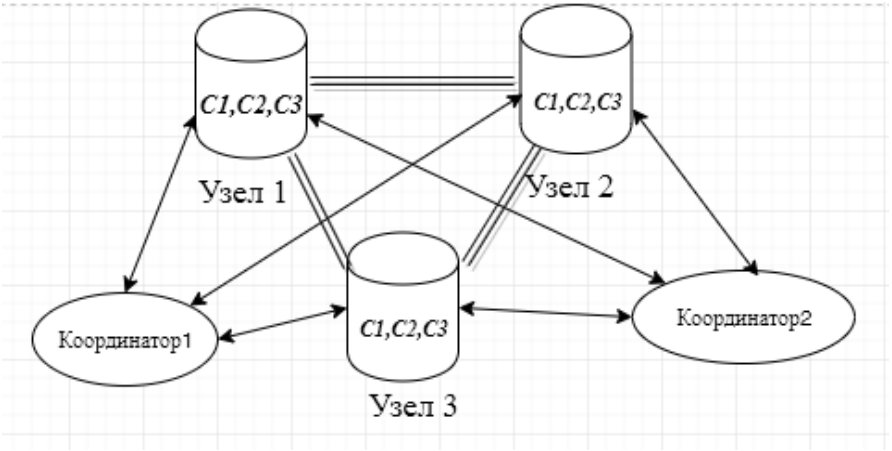


Рис. 6.4. Репликация агрегатов

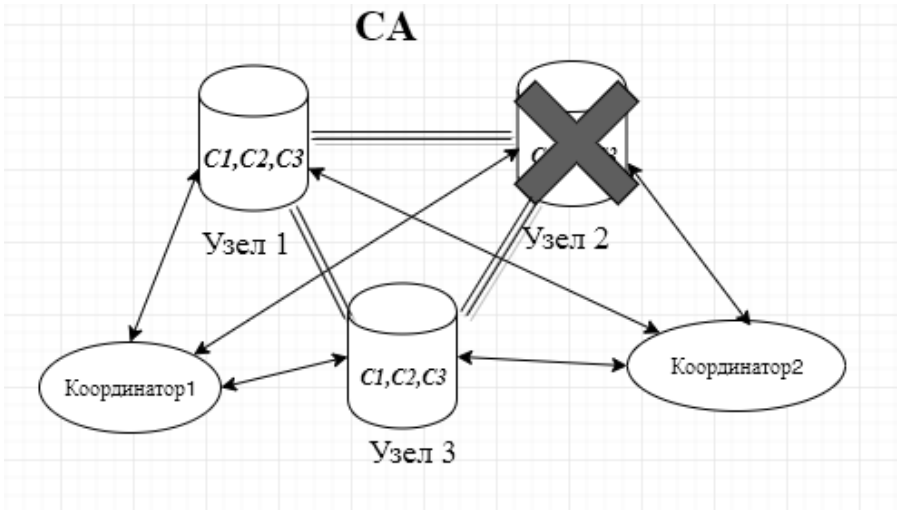


Рис. 6.5. Выполнение свойств C и A

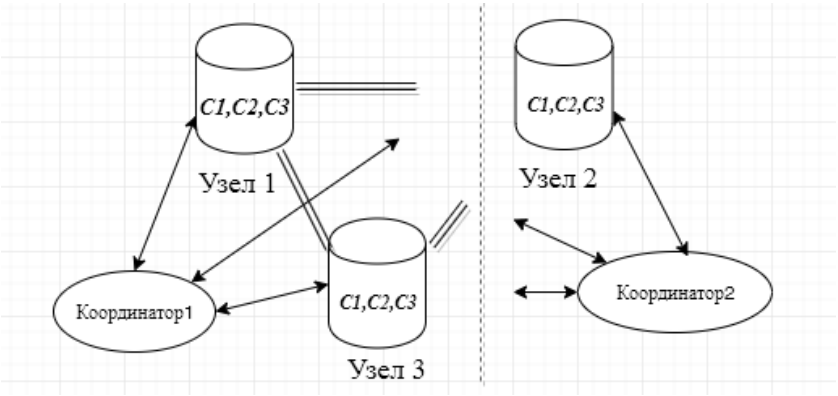


Рис. 6.6. Разделение сети

При одновременной модификации агрегатов и распространении реплик конфликт разрешается с помощью так называемых векторных часов.

В этой ситуации возможно два варианта:

- СУБД перестает работать до восстановления связности;
- СУБД продолжает работать, на разных узлах могут находиться разные версии одного и того же агрегата, как показано на рис. 6.7.

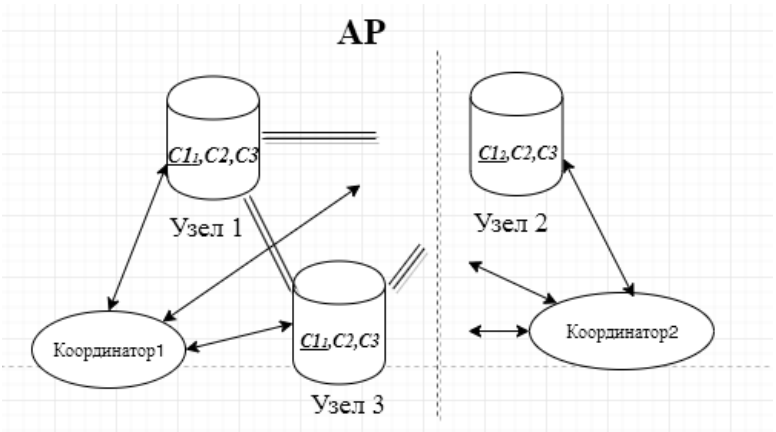


Рис. 6.7. Выполнение свойств A и P

Согласованность данных между узлами нарушится. В зависимости от того, на какой из координаторов придет запрос агрегата C1, вернется либо версия C11, либо C12. После восстановления связности на все узлы распространяются реплики последней версии агрегата.

Перед подробным рассмотрением NoSQL моделей данных подведем предварительный итог их назначения и причин их появления. Эти модели данных и СУБД, их реализующие, не призваны заменить реляционную модель данных для представления предметной области в информационных системах, а предназначены для использования в тех случаях, когда применение реляционных СУБД не может обеспечить необходимой производительности для ряда операций или в конкретных условиях.

6.2. Модель ключ-значение

Структура данных в этой модели представляет собой ассоциативный массив с двумя основными операциями.

- Операция получения значения по ключу. Если ключ не существует, то будет возвращено NULL-значение.
- Запись значения по ключу. Удаление ключа можно рассматривать как вариант этой операции, когда устанавливается NULL-значение. Важная возможность – задание времени существования ключа. По истечении этого времени значение ключ будет удален. Также к этой операции относится модификация значения по ключу.

Эти операции характеризуются константным временем выполнения, не зависящим от объема коллекции. Кроме бинарного объекта, в REDIS могут быть записаны по ключу – список, ассоциативный массив, содержащий объекты. В этом случае операция модификации значения по ключу может изменять отдельные элементы списка или ассоциативного массива, являющегося значением. Из-за наличия всего двух базовых операций (запись объекта и получение объекта) использовать модель данных ключ-значение для представления предметной области в информационной системе невозможно. Однако такая база данных может использоваться как вспомогательная для решения задач кэширования, хранения данных сессии web-приложения, подсчета событий (например, «лайков» записей в социальных сетях). Достоинством выступает простота реализации таких операций. Рассмотрим

фрагмент приложения, где выполняется запись значения по ключу с ограниченным временем жизни и затем его получение.

Реализация этой функциональности в реляционной СУБД была бы следующей.

- Таблица для хранения объектов:
- Необходимо создать индекс для совокупности полей key и add_time.
- Процедура записи данных объекта.
- Процедура получения объекта.

В случае необходимости сохранять список или ассоциативный массив (с обеспечением возможности доступа по ключу к отдельному его элементу) реализация в реляционной СУБД значительно усложнится.

Рассмотрим решение задачи кэширования результатов выполнения запроса к реляционной СУБД. Такая задача возникает при достижении определенного порога запросов к информационной системе. В реляционной СУБД выполняется кэширование блоков данных в оперативной памяти, но при поступлении очередного SQL-запроса требуется его выполнение, даже если в другом сеансе выполнялся такой же запрос с такими же параметрами. Таким образом, одновременное большое количество запросов к реляционной СУБД приведет к проблемам с производительностью, не решаемым в рамках вертикального масштабирования (увеличения количества процессоров и оперативной памяти). Возможно решение методом использования **материализованных представлений** на стороне СУБД. Материализованное представление – это сохраняемый и обновляемый по определенному распорядку результат некоторого запроса. Однако такой метод имеет ряд недостатков:

- не все запросы могут быть оптимизированы таким образом;
- при необходимости изменения запроса может потребоваться изменение материализованного представления;
- обновление большого числа материализованных представлений само по себе может создавать дополнительную нагрузку на СУБД.

При применении «вспомогательной» базы данных ключ-значение кэширование на промежуточном слое (например, на уровне web-сервера) может быть организовано следующим образом – «стандартный» сценарий работы с СУБД: 1) *получение пользовательского запроса*;

2) *формирование SQL-запроса*; 3) *выполнение SQL-запроса и получение данных результата*; 4) *возврат ответа на основе данных из п.4.*

Заменяем на следующую последовательность действий:

1) получение пользовательского запроса;

2) формирование SQL-запроса;

3) формирование ключа из SQL-запроса и параметров;

4) проверка наличия в кэширующей базе данных значения по ключу;

5) если значение найдено – возврат ответа на основе этих данных, если значение не найдено:

5.1) выполнение SQL-запроса и получение данных результата;

5.2) запись в кэширующую базу данных результата из п. 5.1 с заданным временем жизни;

5.3) возврат ответа на основе данных результата из п. 5.2.

Реализация данного метода приводит к усложнению логики информационной системы на слое web-сервера, но обладает следующими достоинствами:

- кэширование выполняется «прозрачно» для основной реляционной СУБД и не требует никаких изменений в самой базе данных;
- кэширование не приводит к дополнительной нагрузке на реляционную СУБД;
- при использовании на слое web-сервера какой-либо библиотеки, инкапсулирующей работу с реляционной СУБД, операции кэширования могут быть реализованы в ней; в этом случае в программном коде, реализующем основную бизнес-логику, нужно будет только задавать параметры, определяющие необходимость использования кэширования при выполнении конкретного запроса, и время жизни результатов.

Основные операции по работе с REDIS при реализации описанного механизма кэширования представлены ниже.

- Формирование ключа по SQL-запросу и его параметрам.
- Запись результата выполнения запроса в коллекцию REDIS.
- Извлечение результата выполнения запроса из коллекции.

Представленный механизм достаточно прост в реализации и эффективен, однако содержит в себе «подводные камни». Предположим, что после модификации данных некоторой сущности все запросы, затрагивающие ее, должны возвращать обновленные данные немедленно после подтверждения транзакции, а не по истечении времени жизни

кэшированных результатов. То есть все данные в кэше, относящиеся к определенному набору SQL-запросов и значений их параметров, должны быть удалены – *инвалидированы*.

Чтобы реализовать такую инвалидацию кэша, потребуются дополнительные коллекции, в которых значениями будут уже ключи основной коллекции-кэша.

Коллекция	Ключ	Значение
Кэш результатов	(SQL-запрос, параметры SQL-запроса)	Результат выполнения SQL-запроса с данными параметрами
Инвалидируемые результаты	(Таблица базы данных, идентификатор строки таблицы)	(SQL-запрос, параметры SQL-запроса)

Изменение алгоритма кэширования для возможности инвалидации результатов, а также непосредственную реализацию кэширования с инвалидацией мы оставляем читателю в качестве самостоятельного упражнения.

Модель ключ-значение может быть расширена введением **упорядоченности** ключей. Такая модель (ordered key-value) позволяет получать значения не по одному ключу (или некоему, перечисляемому множеству ключей), а по диапазону, что может быть эффективно, когда ключом выступает временная метка.

6.3. Документная модель данных

Документная модель может рассматриваться как существенное расширение модели «ключ-значение». В базах данных, реализующих эту модель, значением определенной коллекции, доступным по ключу, является **документ** – некий объект, имеющий свою внутреннюю структуру, элементы, которой доступны на уровне самой базы данных и могут быть аргументами операций, поддерживаемых СУБД. Структура объекта представляет собой ассоциативный массив, ключами которого являются строки, а значениями могут быть атомарные значения (строка, число, байтовый массив, ссылка), другие ассоциативные мас-

сивы и списки. Максимальный уровень вложенности ограничивается конкретной реализацией документной СУБД, так, в текущей реализации MongoDB допустима вложенность до 1000 элементов.

Представим предметную область информационной системы, описывающей учебный процесс вуза. При использовании документной модели данных, например, личное дело студента можно представить следующим документом:

{'Код студента': '0112560012',
'ФИО': 'Иванов Иван Иванович',
'Группа': 'ПМ-81',
'Ссылка на группу': ключ в коллекции групп,
'Успеваемость': [{'Предмет': 'Математический анализ', 'Оценка': 5},...]
],
'Приказы': [{'Номер приказа': '...', 'Дата регистрации': '...' }, {'Номер приказа': '...', 'Дата регистрации': '...' }
]
}

Для приказа следующий документ:

{
'Номер приказа': '...',
'Дата регистрации': '...',
'Пункты приказа': [
'Номер пункта': '...',
'Текст пункта': '...'
]
}

Для группы:

{'Наименование группы': '...',
'Дата формирования': '...',
'Студенты группы': [
{'ФИО': '...', 'Ссылка на студента': '...'}, ...
]
}

В приведенном примере представлен один из возможных вариантов представления данных в коллекциях. Возможны два «предельных» способа представления: «максимально» нормализованный и «максимально» денормализованный.

При нормализованном представлении элементы документов в коллекциях соответствуют полям таблиц в реляционной базе данных, приведенных к третьей нормальной форме. При таком подходе теряется сам смысл использования документной модели данных, так как для получения данных, которые могут быть использованы в приложении, потребуется выполнять соединения коллекций. Эта операция хоть и возможна, но гораздо менее эффективна, чем получение всего документа по ключу, и приводит к накоплению тяжело сопровождаемого программного кода.

При «максимальной» денормализации каждый из документов включал бы в себя данные связанных с ним признаков. Каждый документ из коллекции «студенты» будет включать список, содержащий все данные связанных с ним приказов. При такой организации данных при обращении к любой из коллекций можно будет получить всю необходимую в каждом конкретном случае информацию, без необходимости выполнения как соединений, так и обращения к другому документу по ссылке. Однако в приложении многократно увеличится объем кода в функциях модификации данных. Эмпирическое правило – структура документа в коллекции должна соответствовать и запросам на получение данных, и запросам на их модификацию. Из названия самой модели следует, что в коллекции должны храниться не отдельные факты из предметной области, а именно данные соответствующие документам. Это может быть документ в его «классическом» представлении либо, например, все данные объекта предметной области, обрабатываемые в рамках одной функциональной единицы (например, экранной формы) приложения.

В первоначальном примере показанная «степень» денормализации в коллекциях могла быть выбрана исходя из того, что в рамках экранной формы для личного дела студента достаточно отобразить название группы, в которой он учится, а для получения более подробной информации выполняется переход в другую экранную форму, предназначенную для работы с группой.

Как и в базах данных ключ-значение, в документных базах данных не налагается строгих ограничений на структуру документа. Для MongoDB единственным ограничением является то, что структура документа должна соответствовать требованиям формата JSON (точнее, его надмножеству – BSON). Так, документ {‘a’, [1,2,3]} не соответствует формату JSON и не будет добавлен, но при этом в одну и ту же коллекцию могут быть успешно добавлены два корректных JSON-документа, с различной структурой, например: {‘a’: ‘string’, ‘b’:[1,2,3,4]} и {‘dict1’: {‘a’:1, ‘b’: 2}, ‘dict2’: {‘c’:3, ‘d’: 4}}.

Отметим, что в отличие от базы данных ключ-значение в MongoDB при добавлении документа ключ явно не задается, а генерируется самой СУБД. Поэтому, если выполняется выбор NoSQL СУБД для реализации системы кэширования, где данные необходимо вставлять для заранее заданных значений ключей, то документная база данных не будет подходящим решением.

При необходимости получить выборку из коллекции документов при указании условия на их значения используется метод **find**, где аргументом также выступает JSON-документ, задающий условие:

db.students.find({‘study_group’: ‘ПМ-81’})

При этом возможно, что в коллекции будут документы, в которых отсутствуют атрибуты, перечисленные в условии поиска, в этом случае применение условия к такому документу вернет ЛОЖЬ (кроме случая, когда в условии явно указано, что данный атрибут должен отсутствовать). Отбор документов по условиям на значения атрибутов приводит к полному сканированию всей коллекции, в том числе на всех узлах базы данных, если она распределена. Для предотвращения полного сканирования может создаваться индекс для определенного атрибута (атрибутов) коллекции. Организация индекса B-tree соответствует организации индекса в реляционной базе данных. При создании индекса его экземпляры создаются на всех узлах базы данных, и, как следствие, модификация документа, хранящегося на одном из узлов, будет затрагивать все узлы базы данных.

Возможность отбора документов по условиям на атрибуты документов приближает документные базы данных к возможностям SQL, но язык для получения данных продолжает быть процедурным.

Операция соединения может быть реализована с помощью вспомогательной коллекции, куда добавляются документы, получаемые из запрашиваемых атрибутов документов соединяемых коллекций:

```
db.students.find({{id_study_group:{$exists:true}}}).forEach(
    function(x){
        x.forming_date = db[x. id_study_group.$ref].findOne(
            {_id:x.region.$id},{ forming_date:1}). forming_date;
        db.res.insert(x);
    });
db.res.find({})
```

Недостаток данного метода очевиден – требуется написание достаточного количества кода, объем которого будет возрастать при добавлении коллекций в операцию соединения.

Кроме того, необходимо помнить, что при распределении документов по большому числу узлов операция соединения будет затрагивать в общем случае все из них, значительно увеличивая время отклика.

Локальность хранения данных в одном документе дает преимущество только тогда, когда требуется получить большие части документа за один раз. Базе данных обычно приходится загружать весь документ целиком, даже если вам нужен только маленький его фрагмент, что в случае больших документов будет неэкономно. При обновлении документа, как правило, необходимо тоже переписать весь документ целиком – легко выполнить «на месте» можно только те изменения, которые не меняют закодированного размера документа. Поэтому в большинстве случаев рекомендуется минимизировать размер документов и избегать увеличивающих этот размер операций записи. Это в свою очередь может требовать выполнения соединений. Указанные ограничения по производительности значительно снижают диапазон случаев, когда документоориентированные базы данных могут оказаться полезны.

В современных реляционных СУБД, в частности в PostgreSQL Professional, реализованы возможность хранения JSON документов в столбцах таблицы и выполнение выборки по условию на элементы этих документов.

6.4. Столбцовая модель данных

Столбцовая модель данных, как и документная, является развитием модели ключ-значение. В ней объектом хранения данных является **семейство столбцов** (аналогично таблице в реляционных базах данных и коллекции в документных). Логическая и физическая структуры хранения данных в семействе столбцов в большом числе интернет-публикаций неверно интерпретируются (отчасти из-за того, что ее путают с поколоночным хранением данных в некоторых реляционных базах данных, например Terradata, отчасти из-за того, что синтаксис операции создания семейства столбцов в СУБД Cassandra, реализующей столбцовую модель, очень близок синтаксису создания таблицы на SQL). Семейство столбцов представляет собой двухуровневый ассоциативный массив.

Впервые такая структура была описана под названием BigTable корпорацией Google [https://ai.google/research/pubs/pub27898].

В семействе столбцов обязательно определяется **первичный ключ,** но его назначение отличается от аналогичного понятия в реляционных базах данных. В таблице реляционной базы данных первичный ключ – это ограничение, накладываемое на некоторый набор столбцов таблицы (как правило, состоящий из одного столбца – идентификатора) и гарантирующее уникальность и непустоту значений в них. В семействе столбцов базы данных Cassandra первичный ключ определяет, во-первых, по значениям какого из столбцов будет выполняться распределение данных между узлами кластера. Эта часть первичного ключа называется *ключом разделения* (partition key). Во-вторых, значения какого столбца (или каких столбцов) будут ключами ассоциативного массива второго уровня. Эта часть первичного ключа называется *кластерным ключом* (cluster key).

Ниже приведены операторы CQL создания семейства столбцов **staff** и заполнения его данными.

```
create table staff(
        department text,
        fio text,
        age int,
```

role text,

primary key (department, fio)

);

insert into staff(department, fio, age, role) values (‘Дирекция’, ‘Иванов Иван Иванович’, 60, ‘Директор’);

insert into staff(department, fio, age, role) values (‘Дирекция’, ‘Петров Петр Петрович’, 50, ‘Советник’);

insert into staff(department, fio, age, role) values (‘Дирекция’, ‘Сидорова Мария Ивановна’, 30, ‘Секретарь’);

insert into staff(department, fio, age, role) values (‘Цех №1’, ‘Кузнецов Иван Иванович’, 44, ‘Начальник’);

insert into staff(department, fio, age, role) values (‘Цех №1’, ‘Соколов Иван Иванович’, 45, ‘Бригадир’);

insert into staff(department, fio, age, role) values (‘Цех №1’, ‘Никифоров Иван Иванович’, 42, ‘Рабочий’);

insert into staff(department, fio, age, role) values (‘Цех №2’, ‘Степанов Иван Иванович’, 52, ‘Начальник’);

insert into staff(department, fio, age, role) values (‘Цех №2’, ‘Иванов Иван Иванович’, 40, ‘Бригадир’);

insert into staff(department, fio, age, role) values (‘Цех №2’, ‘Иванов Иван Иванович’, 35, ‘Рабочий’);

Эти данные могут быть выбраны CQL-оператором

*select * from staff*,

который вернет девять строк. Несмотря на внешнюю схожесть синтаксиса приведенных операторов с SQL, организация хранения данных кардинально отличается, от таковой в аналогичной по структуре таблице в реляционной базе данных. В приведенном примере в семействе столбцов **staff** хранится не девять строк, как можно было бы подумать, а три со значениями ключа разделения **department** – ‘Дирекция’, ‘Цех №1’ и ‘Цех №2’. Структура строки ‘Дирекция’ будет включать столбцы ‘Иванов Иван Иванович:age’, ‘Иванов Иван Иванович|role’, ‘Петров

Петр Петрович|age', 'Петров Петр Петрович|role', и так далее. Первую часть наименования столбца формирует значение кластерного ключа, а вторую – собственно наименование атрибута, не входящего в первичный ключ. При добавлении очередной записи, с новым значением кластерного ключа и уже существующим значением ключа распределения, в соответствующей строке произойдет увеличение числа столбцов. Следует иметь в виду, что для семейства столбцов операции **insert** и **update** аналогичны. Так, если после выполнения

insert into staff(department, fio, age, role) values ('Цех №2', 'Иванов Иван Иванович', 35, 'Рабочий');

выполнить вставку записи с таким же значением первичного ключа, но другими значениями неключевых атрибутов

insert into staff(department, fio, age, role) values ('Цех №2', 'Иванов Иван Иванович', 36, 'Инженер');

то результат будет эквивалентен выполнению операции

update staff set age = 36, role= 'Инженер' where department = 'Цех №2' and fio = 'Иванов Иван Иванович'.

При запросе на выборку данных с указанием условия на значение первичного ключа скорость доступа к данным не будет зависеть от общего объема сохраненных в семействе столбцов данных. Запросы на выборку данных из семейства столбцов, если содержатся условия на неключевые атрибуты, должны обязательно включать условие на значение первичного ключа. Данное ограничение предотвращает полное сканирование всех данных в семействе столбцов.

Рассмотрим применение столбцовой модели и основные операции с семействами столбцов на следующем примере. Необходимо сохранять показания, поступающие с различных устройств, для последующей обработки. В один момент времени с каждого устройства поступают значения для нескольких параметров. Количество устройств может достигать нескольких тысяч, а общее количество сохраняемых наблюдений для одного устройства – нескольких десятков миллионов.

При использовании реляционной модели данных ER-диаграмма будет иметь вид, представленный на рис. 6.9.

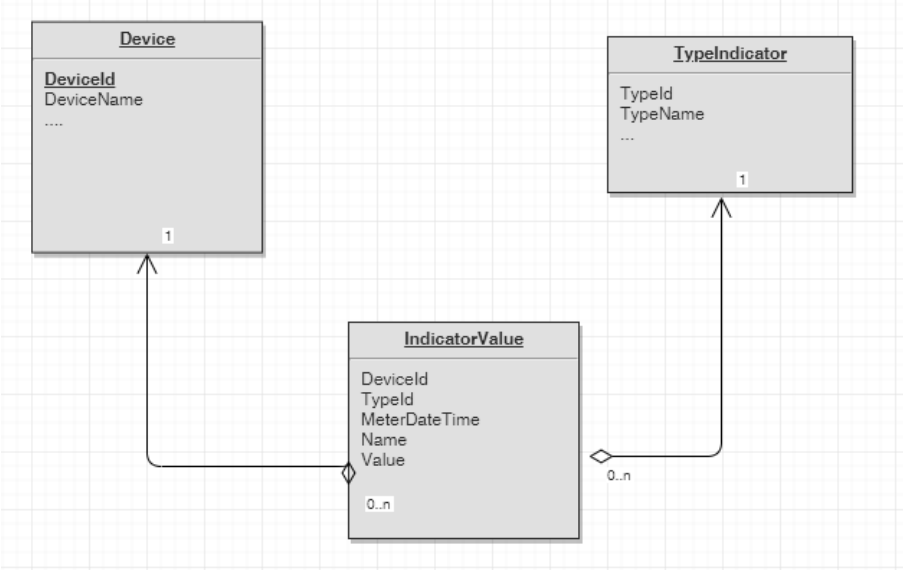


Рис. 6.9. ER-диаграмма реляционной модели данных

Эта структура может быть реализована в любой реляционной базе данных, но при достижении определенного объема наблюдений возникнут проблемы с производительностью.

Реализация в столбцовой СУБД Cassandra позволит распределить данные по произвольному числу узлов, так что данные на одном узле будут полностью умещаться в оперативной памяти узла. Однако так как в столбцовой модели отсутствует операция соединения семейств столбцов, необходимо задать структуру семейства столбцов (или нескольких семейств) исходя из запросов, на выборку, которые будут выполняться к реализуемой базе данных.

Пусть в приложении необходимы следующие запросы:

1) получение для определенного устройства (устройств) выполненных измерений за определенный интервал времени;

2) получение измерений со всех устройств за определенный интервал времени.

Каждый из запросов будет эффективно выполняться при соблюдении следующего условия: в секцию *WHERE* запроса должны входить

столбцы распределительного ключа и (как минимум) ведущие столбцы кластерного ключа.

Заметим, что в текущих реализациях СУБД Cassandra возможно создание вторичного индекса на столбцы, не входящие в первичный ключ и включение в условие отбора этих столбцов. Но создание такого индекса приводит к значительным накладным расходам, так как он в свою очередь должен быть распределен по всем узлам кластера. Кроме того, выборка с использованием такого индекса не гарантирует, что все данные будут получены из одного узла.

Так, для первого запроса будет создано семейство столбцов

```
CREATE TABLE DeviceMeters(
    DeviceName text,
    MeterDateTime Date,
    TypeIndicator text,
    Value text,
    primary key (DeviceName, MeterDateTime)
)
```

Распределение по узлам кластера будет осуществляться в соответствии с наименованием устройства, а время измерения будет кластерным ключом.

Запрос будет иметь вид

```
SELECT *
FROM DeviceMeters
WHERE DeviceName = ‘Device01’  and
      MeterDateTime between ’01-01-2018 00:00:00’ and ‘05-01-2018 00:00:00’;
```

Для второго запроса создается семейство столбцов:

```
CREATE TABLE MetersIndicators(
    MeterDateTime Date,
    TypeIndicator text,
```

DeviceName text,

Value text,

primary key (MeterDateTime, TypeIndicator)

)

В данном случае, наоборот, распределительным ключом будут моменты времени, а запрос будет иметь вид

*SELECT **

FROM MetersIndicators

WHERE MeterDateTime between '01-01-2018 00:00:00' and '05-01-2018 00:00:00';

При разработке приложения нужно учитывать, что данные должны вставляться во все созданные для выполнения запросов семейства столбцов.

Учитывая приведенные достоинства и ограничения столбцовой модели, базы данных на ее основе оправданно применять в приложениях, где важна как высокая скорость выборки данных, так и высокая скорость их вставки, при том что запросы на выборку заранее определены и их количество ограничено.

6.5. Графовая модель данных

Ряд прикладных задач может приводить к необходимости реализации алгоритмов на графах, например:

- социальные графы. Вершины – люди, а ребра отражают знакомство людей друг с другом;
- веб-графы. Вершины – веб-страницы, а ребра отражают HTML-ссылки на другие страницы;
- дороги или железнодорожные сети. Вершины – перекрестки (узловые станции), а ребра соответствуют пролегающим между ними дорогам или железнодорожным линиям.

Граф может быть представлен в реляционной базе данных с помощью двух таблиц – вершин (vertex) и ребер (edge). Ниже приведены операторы создания таблиц и заполнения их данными:

```
create table vertex(
  name varchar2(15) primary key
)
create table edge(
  vertex_in varchar2(15) references vertex(name) not null,
  vertex_out varchar2(15) references vertex(name) not null,
  weight number,
  primary key (vertex_in, vertex_out)
)
begin
  insert into vertex(name) values ('A');
  insert into vertex(name) values ('B');
  insert into vertex(name) values ('C');
  insert into vertex(name) values ('D');
  insert into vertex(name) values ('E');
  insert into vertex(name) values ('F');
end;
begin
  insert into edge values ('A','B',1);
  insert into edge values ('B','C',1);
  insert into edge values ('B','D',2);
  insert into edge values ('C','E',4);
  insert into edge values ('D','E',2);
  insert into edge values ('C','D',10);
  insert into edge values ('E','F',1);
  insert into edge values ('E','B',1);
end;
```

Для поиска всех маршрутов из вершины A в вершину F могут использоваться рекурсивные обобщенные табличные выражения (описаны в стандарте SQL:1999).

```
with a(vi,vo,w,route,total_w) as (
  select e.vertex_in as vi, e.vertex_out as vo, e.weight as w,
e.vertex_in||':'||e.vertex_out route,e.weight total_w
  from edge e
  where e.vertex_in = 'A'
  union all
  select e.vertex_in as vi, e.vertex_out as vo, e.weight as
w,a.route||':'||e.vertex_out  route,a.total_w+e.weight total_w
  from edge e
  join a on e.vertex_in = a.vo
  and not a.route like  '%:'||e.vertex_out||':%'
)
select a.route,a.total_w
from a
where a.vo = 'F';
```

В результате будут получены три маршрута, с указанием веса каждого:

```
A:B:D:E:F      6
A:B:C:E:F      7
A:B:C:D:E:F    15
```

Поиск маршрута с минимальным весом из вершины A в вершину F может быть реализован SQL-запросом с использованием аналитической функции **first_value**, приведенным ниже:

```
with a(vi,vo,w,route,total_w) as (
  select e.vertex_in as vi, e.vertex_out as vo, e.weight as w,
e.vertex_in||':'||e.vertex_out route,e.weight total_w
  from edge e
  where e.vertex_in = 'A'
  union all
  select e.vertex_in as vi, e.vertex_out as vo, e.weight as
w,a.route||':'||e.vertex_out  route,a.total_w+e.weight total_w
```

```
    from edge e
    join a on e.vertex_in = a.vo
)
select distinct
      first_value(a.route) over (order  by total_w) route,
      min(a.total_w) over () total_w
from a
where a.vo = 'F';
```

Результат: *A:B:D:E:F 6*

Следует отметить достаточную сложность обоих запросов, кроме того, выполнение запроса на графе из шести вершин требует 12 обращений к блокам данных (таблиц и индексов). Для графа, состоящего из 20 вершин и 19 ребер, выполнение такого запроса вызовет 201 обращение к блокам таблиц и индексов.

Помимо явного представления графа в реляционной базе данных, необходимость решения задач на графах может возникать для «традиционных» структур данных. Рассмотрим отношения, которые могут связывать студентов и преподавателей. Преподаватель может:

- вести занятия в группе, где учится студент;
- быть тьютором группы, где учится студент;
- выставить оценку студенту;
- быть руководителем проекта (курсового, исследовательского и т.д.), в котором участвует студент;
- быть соавтором публикации, с участием студента.

Каждое из этих отношений реализуется как минимум одной таблицей в реляционной базе данных. Для реализации запроса, возвращающего строки с информацией о том, что связывает конкретного студента и конкретного преподавателя, потребуется включать в запрос соединения со всеми этими таблицами. Хотя данный запрос не будет рекурсивным, но время отклика может также оказаться значительным. Кроме того, при появлении в схеме базы данных таблиц, реализующих новые связи, потребуется изменение этого запроса для включения в него этих таблиц.

Графовые модели данных можно разделить на следующие основные классы:

- модель с метками и свойствами;
- тройные кортежи;
- гиперграфы.

Графовая модель с метками и свойствами

В модели графов свойств каждая вершина состоит:

- из уникального идентификатора;
- множества исходящих ребер;
- множества входящих ребер;
- коллекции свойств (пар «ключ-значение»).

Каждое ребро состоит:

- из уникального идентификатора;
- вершины, с которой оно начинается (*начальная вершина*);
- вершины, которой оно заканчивается (*конечная вершина*);
- коллекции свойств (пар «ключ-значение»).

Некоторые важные аспекты этой модели представлены ниже.

1. Любая вершина может быть соединена ребром с любой другой вершиной. Схема не накладывает ограничения на то, какие элементы могут быть связаны.

2. Для любой вершины можно найти входящие в нее и исходящие из нее ребра и таким образом найти путь по цепочке вершин – как в прямом, так и обратном направлении.

3. Используя различные метки для разных видов связей, можно хранить в одном графе несколько разных аспектов данных моделируемой предметной области.

Рассмотрим реализацию графовой модели с метками и свойствами в СУБД Neo4j и применение декларативного языка Cypher для работы с представлением графа.

1. Создание элементов графа CREATE

Студент Иванов Иван Иванович учится на бюджетной основе в группе ПМ-81 (очной)

create (x:Student{fio:"Иванов Иван Иванович"})-[:STUDY_IN {basis:"Бюджет"}]->(y:Group{name:"ПМ-81",form:"Очная"})

Переменными x и y обозначаются ссылки на вершины, для использования при добавлении новых ребер в рамках того же оператора create. Так, для занесения информации о нескольких студентах, обучающихся в одной и той же группе, у участвующих в некотором проекте, оператор create будет иметь вид:

CREATE
(x1:Student{fio:"Иванов Иван Иванович"})-[:STUDY_IN {basis:"Бюджет"}]
->(y:Group{name:"ПМ-81",form:"Очная"}),
(x2:Student{fio:"Петров Петр Петрович"})-[:STUDY_IN {basis:"Контракт"}]->(y),
(x1)->[:PARTICIPATES]->(p:Project{name:"Исследование алгоритмов сортировки"}),
(x2)->[:PARTICIPATES]->(p);

За пределами оператора create эти переменные не определены.

2. Выборка элементов графа

Для выборки данных служит оператор **MATCH**. Он так же, как и оператор **SELECT** в реляционных базах данных, задает декларативный способ получения данных, но помимо указания условий на значения атрибутов вершин и ребер графа, позволяет задать шаблон, которому должны удовлетворять все выбираемые подграфы.

Оператор MATCH может использоваться, например, для того, чтобы вернуть все ребра с заданной меткой:

MATCH ()-[r:LINK2]->() RETURN r

Вернуть все связанные вершины, где вершина, с которой начинается маршрут, имеет метку Test:

*MATCH (x:Test)-[*1..100]->(y) return x,y*

Для возврата не отдельных вершин, а маршрутов используется следующая форма оператора:

*MATCH p=(x:Test)-[*1..100]->(y) return p*

При необходимости в операторе MATCH могут накладываться условия на значения атрибутов, хранящихся в вершинах и ребрах

*MATCH (x:Test {attr: 'Value'})-[*1..100]->(y) return x,y*

Также возможно задать условие на существование атрибута:

MATCH (n) WHERE EXISTS(n.p1) RETURN n.p1

Рассмотрим реализацию рекомендательной системы для музыкальных записей в социальной сети.

```
create (john:User{name:'John'}),
       (mary:User{name:'Mary'}),
       (piter:User{name:'Piter'}),
       (anna:User{name:'Anna'}),
  (jor:User{name:'Joe'}),
       (ys:Song{name:'Yellow Submarine'}),
       (aynl:Song{name:'All You Need is Love'}),
       (smg:Song{name:'The Show Must Go On'}),
       (ps:Song{name:'People are Strange'}),
       (pib:Song{name:'Paint It Black'}),
  (tm:Song{name:'The Miracle'}),
  (hy:Song{name:'Hey You'}),
       (john)-[:Friend]->(mary),
       (john)-[:Friend]->(piter),
       (piter)-[:Friend]->(joe),
       (john)-[:Listen]->(aynl),
  (john)-[:Listen]->(hy),
       (aynl)-[:Listen]->(mary),
       (mary)-[:Listen]->(ys),
       (aynl)-[:Listen]->(anna),
       (anna)-[:Listen]->(smg),
       (piter)-[:Listen]->(ps),
       (piter)-[:Listen]->(pib)
```

Задаваемый этими данными граф представлен на рис. 6.10.

Вершинами являются пользователи и песни, ребра реализуют связи между пользователями: пользователь X является другом пользователя Y, и между пользователями и песнями – пользователь X слушает песню Z.

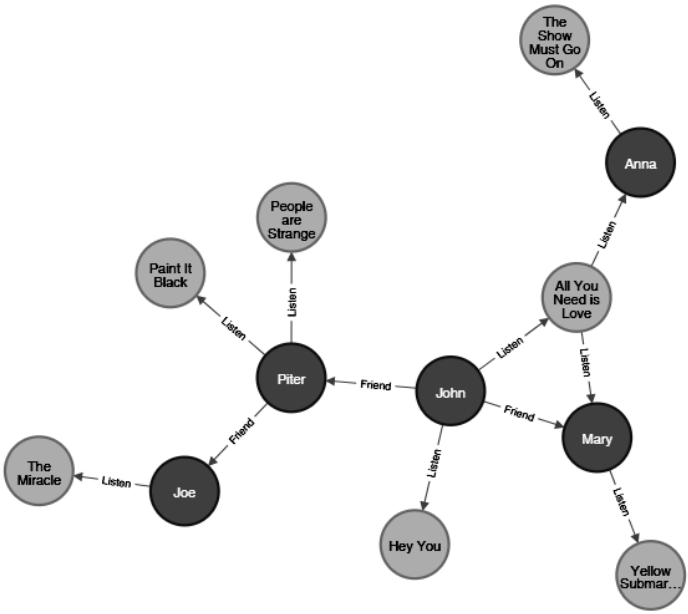


Рис. 6.10. Граф пользователей социальной сети и песен, которые они слушают

Пусть требуется для заданного пользователя вернуть те песни, которые слушают его друзья, а также те песни, которые слушают пользователи, не являющиеся его друзьями, но слушающие те же песни, что и он сам. Для того чтобы получить, какие песни слушают друзья пользователя с именем **John**, используется оператор:

match(x:User{name:'John'})-[:Friend]->()-[:Listen]->(y:Song) return y.

Следующий оператор позволяет получить песни, которые слушают пользователи, слушающие то же, что и **John**:

match(x:User{name:'John'})-[:Listen]->()-[:Listen]->()-[:Listen]->(y:Song) return y.

или более кратко

*match(x:User{name:'John'})-[:Listen*3]->(y:Song) return y.*

Для объединения результатов используется операция union, так же, как и в SQL-запросах:

*match(x:User{name:'John'})-[:Listen*3]->(y:Song) return y.*

union

*match(x: User {name: 'John'})-[:Listen*3]->(y: Song) return y.*

Можно получить не только песни, но и количество маршрутов от выделенного пользователя до каждой них, что позволит установить приоритетность песен:

*match(x: User {name: 'John'})-[:Listen*3]->(y: Song)*

with x, y

match p=(x)-[*]->(y)

return y, count(*)

union

match(x: User {name: 'John'})-[:Friend]->(:User)-[:Listen]->(y: Song)

with x, y

match p=(x)-[*]->(y)

return y, count(*)

В итоге мы получили все маршруты от пользователя **John** до найденных песен и подсчитали их количество.

При графовом моделировании объектов предметной области необходимо соблюдать аккуратность при выборе использования вершин или ребер для представления определенных фактов. Так, ребро может связывать только две вершины. Поэтому, если в ER-модели предметной области существует связь многие-ко-многим между более чем

двумя сущностями, обладающая при этом своими атрибутами, то реализация ее в виде ребер между вершинами, соответствующими сущностям, приведет к тому, что данные атрибутов будут дублироваться в каждом из ребер. Целесообразнее представлять такую связь тоже в виде вершин, связанных ребрами с вершинами, представляющими соответствующие сущности.

Например, для моделирования ситуации «Преподаватель поставил студенту оценку по предмету» представление оценки как набора ребер между вершинами «Преподаватель», «Студент», «Дисциплина» приведет к тому, что в каждом из трех ребер помимо дублирования значения оценки нужно будет хранить идентификатор несвязанной этим ребром вершины, представляющей участвующий в отношении объект.

Для соблюдения логической согласованности в графовой базе данных Neo4J можно налагать ограничения на значения атрибутов узлов и связей.

Оператор *CREATE CONSTRAINT ON (t:Test) ASSERT t.p1 IS UNIQUE* устанавливает ограничение уникальности значений атрибута p1 для всех вершин с меткой **Test**.

Оператор *CREATE CONSTRAINT ON (book:Book) ASSERT exists(book.isbn)* устанавливает требование существования значений атрибута **isbn** для всех вершин с меткой **Book**.

Для связей может устанавливаться требование существования значений. Оператор *CREATE CONSTRAINT ON ()-[like:LIKED]-() ASSERT exists(like.day)* устанавливает требование существования значений атрибута **day** для всех связей с меткой LIKED.

Хранилища тройных кортежей

Модель тройных кортежей практически эквивалентна модели графов свойств и использует другие понятия для описания одних и тех же идей.

В хранилище тройных кортежей вся информация хранится в форме трехкомпонентных высказываний: (*субъект, предикат, объект*). Например, в тройном кортеже (*Иванов, учится, ПМ-81*) Иванов – субъект (подлежащее), учится – предикат (сказуемое), ПМ-81 – объект (дополнение).

Субъект тройного кортежа эквивалентен вершине графа. Объект представляет собой один из двух вариантов.

1. Значение простого типа данных, например строчного или числового. В этом случае предикат и объект тройного кортежа эквивалентны ключу и значению свойства вершины-субъекта. Например, (*Иванов, возраст, 33*) эквивалентно вершине *Иванов* со свойствами {"Возраст":33}.

2. Другую вершину графа. В этом случае предикат представляет собой ребро графа, субъект – начальную вершину, а объект – конечную вершину. Например, в приведенном выше тройном кортеже (*Иванов, учится, ПМ-81*) субъект и объект – *Иванов* и *ПМ-81* – оба представляют собой вершины, а предикат *учится* – метку на соединяющем их ребре.

На базе модели тройных кортежей была разработана модель RDF – Resource Description Framework (среда описания ресурса). Данные RDF могут быть записаны в формате XML:

```
<rdf:RDF xmlns="urn:example:"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<Location rdf:nodeID="idaho">
<name>Idaho</name>
<type>state</type>
<within>
<Location rdf:nodeID="usa">
<name>United States</name>
<type>country</type>
<within>
<Location rdf:nodeID="namerica">
<name>North America</name>
<type>continent</type>
</Location>
</within>
</Location>
</within>
```

</Location>

<Person rdf:nodeID="lucy">

<name>Lucy</name>

<bornIn rdf:nodeID="idaho"/>

</Person>

</rdf:RDF>

У RDF есть несколько особенностей в силу его нацеленности на обмен данными в масштабе всего Интернета. Субъект, предикат и объект тройного кортежа часто представляют собой URI. Например, предикат может быть вот таким URI: <http://my-company.com/namespace#within>, а не просто WITHIN или LIVES_IN. URL http://my-company.com/namespace, не обязательно должен разрешаться – с точки зрения модели RDF это просто пространство имен

SPARQL – язык запросов для хранилищ тройных кортежей, использующих модель данных RDF (это аббревиатура для *SPARQL Protocol and RDF Query Language* (Протокол SPARQL и язык запросов RDF)). Он предшествовал Cypher и концепция поиска по шаблону языка Cypher заимствована из него. Пример запроса на языке SPARQL для поиска людей, эмигрировавших из США в Европу, для представленной выше записи:

PREFIX : <urn:example:>

SELECT ?personName WHERE {

?person :name ?personName.

?person :bornIn / :within / :name "United States".*

?person :livesIn / :within / :name "Europe".*

}

Гиперграфы

Гиперграф является расширением понятия ребра в том случае, когда одно ребро может одновременно соединять более двух вершин. Таким образом, в гиперграфовой модели данных ребро может реализовать связь между более чем двумя сущностями, что невозможно в обычной графовой модели. Недостатком гиперграфовой модели можно считать сложность эффективной реализации выполнения запросов в таких базах данных.

Графовые базы данных имеют общие признаки с базами данных, реализующих сетевую модель, однако между ними есть ряд принципиальных различий, не позволяющих считать их неким частным случаем сетевой модели.

У сетевых баз данных есть схема, которая определяет, какие типы записей могут быть вложены в записи других типов. В графовых базах данных любая вершина может быть соединена ребром с любой другой. Это позволяет приложениям гораздо более гибко адаптироваться к меняющимся требованиям к хранимым и обрабатываемым данным.

Единственный способ получить конкретную запись в сетевой базе данных – пройти по одному из путей доступа к ней. В графовой базе данных можно сослаться непосредственно на любую вершину по ее уникальному идентификатору или воспользоваться индексом для поиска вершины с конкретным значением.

В сетевой модели данных все потомки конкретной записи были упорядоченным множеством, поэтому в СУБД нужно было поддерживать это упорядочение (что оказывало свое влияние на структуру хранилища), а приложениям при вставке новых записей необходимо было обеспечивать соответствующее расположение записей в этих множествах. В графовой базе данных вершины и ребра не упорядочены (можно отсортировать только результаты выполнения запроса).

В сетевой модели данных все запросы были императивны и могли быть нарушены при внесении изменений в схему. Большинство графовых баз данных поддерживает высокоуровневые декларативные языки запросов, такие как Cypher или SPARQL.

6.6. Общие замечания по NoSQL моделям данных

Можно выделить несколько основных причин широкого внедрения баз данных NoSQL, включая:

- потребность в бóльших возможностях масштабирования, чем у реляционных баз данных, включая обработку очень больших наборов данных или очень большую пропускную способность по записи;
- предпочтение свободного программного обеспечения вместо коммерческих продуктов;
- специализированные запросные операции, плохо поддерживаемые реляционной моделью;

- стремление к более динамичным и выразительным моделям данных.

Однако каждая из NoSQL моделей вместе с возможностями по эффективной обработке большого объема данных несет в себе существенные ограничения по применимости. В итоге у каждого приложения имеются свои требования, и оптимальная технология может различаться в зависимости от сценария использования. В ближайшем будущем реляционные базы данных будут продолжать задействовать возможности разнообразных нереляционных баз данных. Такой способ называется *применение нескольких систем хранения данных в одном приложении* (polyglot persistence).

Также в настоящее время разрабатываются новые системы управления базами данных, категория которых носит общее название NewSQL и характеризуется следующими особенностями:

- SQL как основной механизм для взаимодействия;
- ACID поддержка транзакций;
- механизм управления без применения блокировок, таким образом считывающие данные в реальном времени не будут находиться в противоречии с записывающими;
- удобное масштабирование, способное управлять большим количеством узлов, не перенося узкие места.

Контрольные вопросы и упражнения

1. Привести основные операции в базах данных ключ-значение.
2. Укажите возможности применения баз данных ключ-значение.
3. В чем преимущества и недостатки документной модели данных?
4. Чем определяется выбор уровня денормализации документа при использовании документных баз данных?
5. Каково основное назначение баз данных на основе семейств столбцов?
6. Каково назначение первичного ключа в СУБД Cassandra?
7. Создайте кластер из баз данных Cassandra на нескольких узлах, сравните скорость вставки и выборки данных в семейство столбцов с таблицей в реляционной СУБД.
8. Перечислите типы графовых моделей данных.
9. В чем сходство и различие операторов MATCH и SELECT?

7. Темпоральные базы данных

При разработке базы данных часто наряду с отношениями, содержащими сведения о текущем состоянии предметной области (сотрудники, детали…), приходится иметь дело с отношениями, отражающими историю их изменения: история закупок деталей (рис. 7.1, *а*), передача деталей в производство (рис. 7.1, *б*), изменение средней цены детали при поступлении новой партии (рис. 7.1, *в*), история продвижения по службе сотрудников (рис. 7.1, *г*).

Принципиально временные запросы могут обрабатываться в рамках обычной реляционной модели. Имея таблицы подобной структуры и соответствующие данные, вы можете далее обрабатывать запросы, ассоциированные со временем, причем некоторые выполнить достаточно легко: например, запрос относительно истории поступления деталей отдельного вида выглядит следующим образом:

select название_детали, цена, дата_введения_цены

from T1 where название_детали = “Болт”

Запрос, требующий нахождения истории изменения средней стоимости детали (или стоимости детали на указанную дату), выполнить будет сложнее, но его тоже можно записать с использованием данных таблиц (T1, T2, T3).

Можно придумать еще более сложные запросы, которые потребуют либо крайне больших усилий, либо изменения структуры базы данных:

- Каково было число деталей и их стоимость на начало каждого года?
- Когда имело место на складе наибольшее количество деталей указанного вида?

• Кто из сотрудников и в какой период времени имел наибольший карьерный рост?

• В какой месяц компания несла наибольшие затраты на заработную плату?

Название_детали	Кол-во	Цена	Дата_закупки	Изготовитель
Болт	300	2.5	01.01.2000	З-д Труд
Болт	400	2.3	30.06.2001	Инструментальный з-д
Винт	200	1.8	15.10.2000	Инструментальный з-д
Болт	400	2.6	10.02.2002	З-д Большевик
…	…	…	…	…

а

Название детали	Выдано	Дата выдачи
Болт	10	10.01.2000
Винт	15	15.06.2001
Болт	5	15.02.2000
Болт	10	10.03.2002
…	…	…

б

Название детали	Текущее кол-во	Средняя_цена текущая
Винт	120	1.9
Болт	350	2.45
…	…	…
	…	…

в

Фамилия_ служащего	Должность	Зарплата	Дата_назначения
Иванов	Программист	300	01.01.2000
Петров	Программист	350	01.01.2000
Иванов	Аналитик	400	01.06.2001
Сидоров	Рук.группы	500	16.02.2000
Иванов	Системный интегратор	500	01.01.2002
Сидоров	Рук.отдела	700	01.01.2002

г

Рис. 7.1. Отношения, содержащие время:

а – история покупок (Т1); *б* – история использования в производстве (Т2); *в* – история изменения цены (Т3); *г* – карьерный рост сотрудников (Т4)

Подобные рассуждения позволяют сделать вывод, что время представляет собой такое измерение, с которым нелегко обходиться в традиционных СУБД. Одно из решений данной проблемы может быть найдено с помощью временных баз данных, когда время встраивается в саму структуру базы данных. Иными словами, пользователю не требуется создавать таблицы с исторической информацией, основанные на заранее предусмотренных запросах, вместо этого соответствующие части базы данных будут автоматически иметь измерение времени.

Основные принципы временных баз данных

1. Расширяется реляционная модель. К двумерным объектам (кортежи, атрибуты) добавляется третье измерение – время, в результате чего мы получаем трехмерное временное отношение (рис. 7.2). При этом время автоматически встраивается в само отношение.

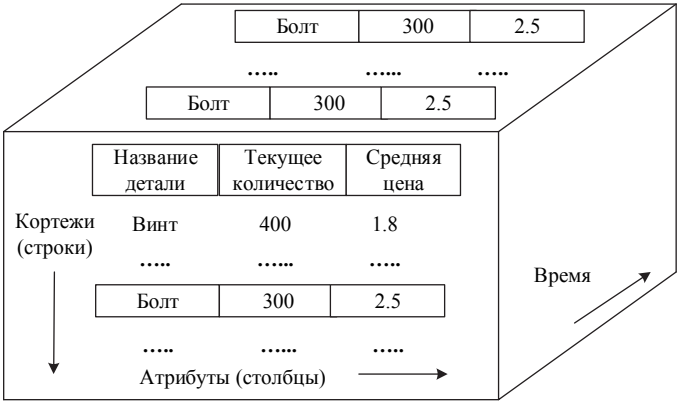


Рис. 7.2. Трехмерное временное отношение

2. Уточняется понятие «время».

Термины	Понятия
Эффективное время (действительное время)	Время, когда факт вступает в силу в действительности
Время регистрации (время транзакции)	Время, когда значение фактически помещается в базу данных
Время, определенное пользователем	Обычный домен времени, не представимый в виде временного куба

Существует несколько экспериментальных временных моделей данных.

1. Temporal Relational Model (TRM)

Временная база данных определяется как объединение двух множеств отношений R_s и R_t, где R_s – множество всех статических отношений, а R_t – множество всех отношений, изменяющихся во времени. Временные отношения включают два обязательных атрибута T_s (the start time) и T_e (the end time) – нижнюю и верхнюю границу временного интервала соответственно. Каждый кортеж отношения, меняющегося во времени (Time-Varying, TVR), обязательно имеет нижнюю границу временного интервала T_s, верхняя граница T_e может быть неизвестной.

При этом T_e может быть бесконечным (деталь поступила и здесь осталась) либо конечным, но пока еще неизвестным (сотрудник принят на работу на конечный, но неопределенный срок). Можно отметить, что TRM использует действительное время (рис. 7.3).

Фамилия служащего	Должность	Зарплата	T_s	T_e
Иванов	Программист	300	01.01.2000	30.05.2001
Петров	Программист	350	01.01.2000	
Иванов	Аналитик	400	01.06.2001	31.12.2001
Сидоров	Рук.группы	500	16.02.2000	31.12.2001
Иванов	Системный интегратор	500	01.01.2002	
Сидоров	Рук.отдела	700	01.01.2002	

Рис. 7.3. Temporal Relational Model (TRM)

В модели TRM используется концепция временной нормальной формы TNF (Time Normal Form): отношение находится в TNF тогда и только тогда, когда оно находится в нормальной форме Бойса–Кодда и не существует каких-либо временных зависимостей между неключевыми атрибутами.

Для работы с моделью TRM используется расширение языка SQL – язык TSQL. Например, для выборки оператор **select** дополнен конструкцией **when**:

select {целевой список} from {список отношений}

where {условия}

when {временное выражение}

Временные операторы сравнения в конструкции **when**:

before (перед);	*during (во время);*
after (после);	*follows (следует за);*
overlap (перекрывает);	*equivalent (равно);*
adjacent (смежный);	*precedes (предшествует).*

Кроме этого, язык TSQL позволяет выполнять выборку отметок времени, выборку информации, упорядоченной по времени, получать временные срезы, группировать по времени.

2. Historical Data Model (HDM)

Данная модель временной базы данных связана с так называемыми периодами жизни, которые ассоциируются с объектами различной гранулярности (рис. 7.4).

Для работы с HDM-моделью также используется расширение языка SQL – язык HSQL. В нем, в частности для определения данных посредством фразы **with time granularity** (с гранулярностью времени), делается указание на то, с каким достаточным уровнем гранулярности заносятся данные:

create table История детали (название детали char(30) not Null,

цена dec(5,2),

количество int)

with time granularity date;

В приведенном примере гранулярность задается с точностью до даты. При необходимости это могли бы быть

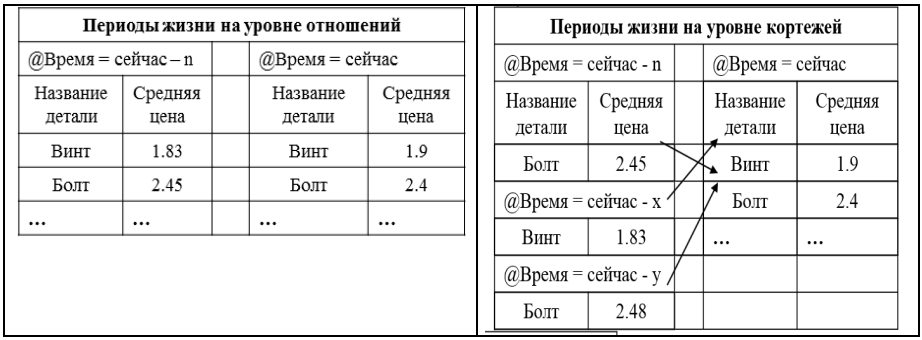
date:hour – день:час

date:hour:min:sec – день:час:мин:сек.

В таблице исторических данных, представленных с помощью приведенного выше синтаксиса, к явно специфицированным столбцам добавляются еще два

Название детали Цена Количество From To

Столбцы **From, To** позволяют фиксировать все временные интервалы с заданным уровнем гранулярности.



Периоды жизни на уровне отношений

@Время = сейчас – n			@Время = сейчас	
Название детали	Средняя цена		Название детали	Средняя цена
Винт	1.83		Винт	1.9
Болт	2.45		Болт	2.4
…	…		…	…

Периоды жизни на уровне кортежей

@Время = сейчас - n			@Время = сейчас	
Название детали	Средняя цена		Название детали	Средняя цена
Болт	2.45		Винт	1.9
@Время = сейчас - x			Болт	2.4
Винт	1.83		…	…
@Время = сейчас - y				
Болт	2.48			

Периоды жизни на уровне атрибутов

@Время = сейчас				
Название детали		Текущее кол-во		Средняя цена
Винт		120		1.9
Болт		350		2.45
…		…		…
@Время = a		@Время = c		@Время = d
Болт 240		Болт 2.42		Болт 2.44
	@Время = b		@Время = e	
	Болт 340		Винт 140	

Рис. 7.4. Historical Data Model (HDM)

HSQL имеет набор фраз для поддержки временных операций при выборке:

- *in interval* в конструкции where;
- *fromtime …totime* (задание интервала);
- *contains* (содержит)

и набор встроенных функций:

- *current()* – представить текущие (последние) кортежи отношения;
- *history()* – представить только исторические кортежи отношения;
- *pred()* – значение времени, которое предшествует временному аргументу в гранулярности этого аргумента.

Контрольные вопросы и упражнения

1. Что такое эффективное время? Время транзакции?
2. Дайте характеристику модели TRN.
3. Каково условие нахождения отношения во временной нормальной форме?
4. Дайте характеристику модели HDM.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Базы данных. Теория и практика: [учебник для вузов по направлению «Информатика и вычислительная техника» и «Информационные системы»] / Б.Я. Советов, В.В. Цехановский, В.Д. Чертовской. – М., 2007.

2. *Хансен Г., Хансен Д.* Базы данных: разработка и управление. – М.: Бином, 1999.

3. Базы данных / А.Д. Хоменко, В.М. Цыганков, М.Г. Мальцев. – СПб.: Корона, 2002.

4. *Рудикова Л.В.* Базы данных: разработка приложений. – СПб.: БХВ-Петербург, 2006.

5. *Кузнецов С.Д.* Базы данных. – М.: Академия, 2012.

6. Базы данных. Проектирование, реализация и сопровождение / Т. Конноли, К. Бегг, А. Страчан. – М. – СПб. – Киев, 1999.

7. *Дейт К.* Введение в системы баз данных. – М.: Вильямс, 2005.

8. Методы и модели анализа данных: OLAP и Data Mining / А.А. Барсегян, М.С. Куприянов, В.В. Степаненко, И.И. Холод. – СПб.: БХВ-Петербург, 2004.

9. Бизнес-аналитика: от данных к знаниям: учебное пособие / Н.Б. Паклин, В.И. Орешков. – СПб.: Питер, 2010.

10. Базы данных. Интеллектуальная обработка информации / В.В. Корнеев, А.Ф. Гарев, С.В. Васютин, В.В. Райх. – М.: Нолидж, 2000.

11. *Клеппман М.* Высоконагруженные приложения. Программирование, масштабирование, поддержка. – СПб.: Питер, 2018. – 640 с.: ил. – (Серия «Бестселлеры O'Reilly»).

12. *Фаулер, Мартин, Садаладж, Прамодкумар Дж.* NoSQL: новая методология разработки нереляционных баз данных: пер. с англ. – М.: Вильямс, 2013. – 192 с.: ил.

13. *Робинсон Ян, Вебер Джим, Эифрем Эмиль*. Графовые базы данных: новые возможности для работы со связанными данными / пер. с англ. Р.Н. Рагимова; науч. ред. А.Н. Кисилев. – 2-е изд. – М.: ДМК Пресс, 2016. – 256 с.: ил.

14. *Бэнкер Кайл*. MongoDB в действии / пер. с англ. А.А. Слинкина. – М.: ДМК Пресс, 2012. – 394 с.: ил.

15. *Редмонд Эрик, Уилсон Джим Р*. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL / под ред. Ж. Картер; пер. с англ. А.А. Слинкин. – М.: ДМК Пресс, 2013. – 384 с.: ил.

16. *Карпентер Д., Хьюитт Э.* Cassandra. Полное руководство. 2-е изд. / пер. с англ. А.А. Слинкин. – М.: ДМК Пресс, 2017. – 400 с.: ил.

Аврунев Олег Евгеньевич
Стасышин Владимир Михайлович

МОДЕЛИ БАЗ ДАННЫХ

Учебное пособие

Выпускающий редактор *И.П. Брованова*
Корректор *И.Е. Семенова*
Дизайн обложки *А.В. Ладыжская*
Компьютерная верстка *С.И. Ткачева*

Налоговая льгота – Общероссийский классификатор продукции
Издание соответствует коду 95 3000 ОК 005-93 (ОКП)

Подписано в печать 06.12.2018. Формат 60 × 84 1/16. Бумага офсетная. Тираж 100 экз.
Уч.-изд. л. 7,2. Печ. л. 7,75. Изд. № 311. Заказ № 142. Цена договорная

Отпечатано в типографии
Новосибирского государственного технического университета
630073, г. Новосибирск, пр. К. Маркса, 20